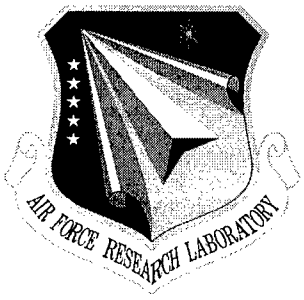
AFRL-IF-RS-TR-2000-10
Final Technical Report
February 2000



BROADCAST OBJECTS FOR EFFECTIVE DATA DISSEMINATION IN BADD

Brown University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F078

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

DTIC QUALITY INSPECTED 3

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-10 has been reviewed and is approved for publication.

APPROVED: William E Rzepka

WILLIAM E. RZEPKA
Project Engineer

FOR THE DIRECTOR: Northrup Fowler

NORTHRUP FOWLER
Technical Advisor
Information Technology Division

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

BROADCAST OBJECTS FOR EFFECTIVE DATA DISSEMINATION IN BADD

Stanley B. Zdonik

Contractor: Brown University
Contract Number: F30602-97-2-0241
Effective Date of Contract: 03 July 1997
Contract Expiration Date: 30 May 1999
Short Title of Work: Broadcast Objects for Effective Data Dissemination in BADD
Period of Work Covered: Jul 97 – May 99

Principal Investigator: Stanley B. Zdonik
Phone: (401) 863-7648
AFRL Project Engineer: William E. Rzepka
Phone: (315) 330-2762

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by William E. Rzepka, AFRL/IFTD, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2000	Final Jul 97 - May 99

4. TITLE AND SUBTITLE
BROADCAST OBJECTS FOR EFFECTIVE DATA DISSEMINATION IN BADD

5. FUNDING NUMBERS
C - F30602-97-2-0241
PE - 63750D
PR - IIST
TA - 00
WU - 14

6. AUTHOR(S)
Stanley B. Zdonik

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Brown University
Computer Science Department
Box 1910
Providence RI 02912

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 N Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFTD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-10

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: William E. Rzepka/IFTD/(315) 330-2762

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Data dissemination systems are difficult to design properly. Since interest in this topic is relatively new, there is little excerption with methods for making performance decisions. This project has been focused on providing tools for understanding the impact of design decisions in dissemination-based data delivery.

We have developed two broad classes of tools. The first is a set of simulation-based tools that allowed us to study fundamental algorithms. We have applied these simulators to the problem of information channelization. We have built a test harness for controlling the deployment of multiple experiments and the collection of results into a convenient graphical form. We see this a major step toward a commander's workbench, a tool to help commanders make resource allocation decisions.

The second class of tool is a toolkit that allows us to quickly assemble prototype (perhaps in a local-area network) that mimics the ultimate deployed system. A prototype of this kind can be instrumented and performance measurements can be collected that gives us more realistic insight than a simulator. This approach provides the second line of defense.

14. SUBJECT TERMS
Data Broadcast, Push Technology, Software, Computers

15. NUMBER OF PAGES
100

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

1 Dissemination-Based Information Systems 2

1.1 The DBIS Framework 2

1.2 The DBIS Toolkit and Example Application 2

2 On-Demand Broadcasting 3

2.1 Scheduling with RxW 3

2.2 Data Staging 4

3 Self-Adaptive User Profiles for Large Scale Data Delivery 5

4 Information Quality Metrics 5

5 Channelization 6

5.1 Simulator Structure 6

5.2 Simulator Measurements 7

6 References 8

Appendix A A Scheduling Approach for Large-Scale On-Demand Data Broadcast

Appendix B Data Staging for On-Demand Broadcast

Appendix C Research in Data Broadcast and Dissemination

Appendix D DBIS-Toolkit: Adaptive Middleware for Large Scale Data Delivery

Appendix E A Framework for Scalable Dissemination-Based Systems

Appendix F Data in Your Face: Push Technology in Perspective

Abstract

Data dissemination systems are difficult to design properly. Since interest in this topic is relatively new, there is little experience with methods for making performance decisions. This project has been focused on providing tools for understanding the impact of design decisions in dissemination-based data delivery.

We have developed two broad classes of tools. The first is a set of simulation-based tools that allowed us to study fundamental algorithms. We have applied these simulators to the problem of information channelization. We have built a test harness for controlling the deployment of multiple experiments and the collection of results into a convenient graphical form. We see this a major step toward a commander's workbench, a tool to help commanders make resource allocation decisions.

The second class of tool is a toolkit that allows us to quickly assemble prototype (perhaps in a local-area network) that mimics the ultimate deployed system. A prototype of this kind can be instrumented and performance measurments can be collected that gives us more realistic insight than a simulator. This approach provides the second line of defense.

Technical Report

1 Dissemination-Based Information Systems

The proliferation of the Internet and intranets, the development of wireless and satellite networks, and the availability of asymmetric, high-bandwidth links to the home, have fueled the development of a wide range of new "dissemination-based" applications. These applications involve the timely distribution of data to a large set of consumers, and include stock and sports tickers, traffic information systems, electronic personalized newspapers, and entertainment delivery. Dissemination-oriented applications have special characteristics that render traditional client-server data management approaches ineffective. These include:

- tremendous scale.
- a high-degree of overlap in user data needs.
- asymmetric data flow from sources to consumers.

To address the particular needs of dissemination-based applications, we have developed a general framework for describing and constructing Dissemination-Based Information Systems (DBIS) [Fran97, Fran98]. The framework incorporates a number of data delivery mechanisms and an architecture for deploying them in a networked environment. The goal is to support a wide range of applications across many varied environments, such as mobile networks, satellite-based systems, and wide-area networks. By combining the various data delivery techniques in a way that matches the characteristics of the application and achieves the most efficient use of the available server and communication resources, the scalability and performance of dissemination-oriented applications can be greatly enhanced. We have constructed an initial version of a toolkit that implements this framework, and have demonstrated it at the 1999 ACM SIGMOD International Conference on the Management of Data [Alti99].

1.1 The DBIS Framework

The basic concepts of the DBIS framework were presented at the OOPSLA 97 conference [Fran97]. A more recent description appears in [Akso98b]. The two major features of the framework are: First, it incorporates a number of different options for data delivery, including traditional request-response, publish/subscribe, Broadcast Disks and on-demand broadcast [Akso98a]. Second, it is based on the notion of *network transparency*, which allows different data delivery mechanisms to be mixed-and-matched within a single application. Network transparency is provided through the use of *Information Brokers*, which acquire information and distribute it to other consumers. Brokers are middlemen; a broker acts as a client to some number of data sources, collects and possibly repackages the data it obtains, and then functions as a data source to other nodes of the system. Along the way, brokers may add value to the information, such as integrating it with data from other sources or enhancing its organizational structure. By creating hierarchies of brokers, information delivery can be tailored to the needs of many different users.

1.2 The DBIS Toolkit and Example Application

We have developed an initial prototype of a toolkit that implements the DBIS architecture. The toolkit is written in Visual C++ and runs on Windows NT. It exploits the IP-Multicast support that is included with NT 4.0. The toolkit consists of 30,000 lines of code, although a portion of this consists of code generated by the Visual C++ tools for user interface functions. The toolkit is described in more detail in [Alti99].

The toolkit provides a set of application programming interfaces (APIs) and libraries that allow a developer to construct and experiment with a DBIS application. The DBIS-Toolkit consists of four main components:

Data Source (DS) Library - a data source wrapper that encapsulates network communication and provides conversion functions for data.

Client Library - a client program wrapper that encapsulates network communication and provides conversion functions for queries and user profiles. It also provides monitoring and filtering of broadcast or multicast channels.

Information Broker (IB) - the main component of the DBIS-Toolkit. The IB contains communication, buffering, scheduling, and catalog management components and is described in more detail below.

Information Broker Master - The IB Master is responsible for managing global catalog information about data and the topology of the DBIS. All IBs must register with the IB Master and all catalog updates must be sent to the IB Master.

In addition to these four components, the toolkit contains a flexible performance monitoring and instrumentation interface that can be used to graphically display real-time performance metrics such as bandwidth and CPU utilization, response times, etc. on a per-IB basis. The instrumentation tool also allows more application-specific metrics to be obtained and displayed.

We have used the toolkit to construct a CONUS weather map dissemination application, in which weather maps for various regions of the Continental United States can be delivered to large numbers of users via push/pull, multicast/unicast and their various combinations. Perhaps most impressively, we demonstrated how the data delivery mechanisms between various components can be changed *on-the-fly*, without requiring the application to be restarted, and without impacting components that were not directly connected to the changed data flow.

2 On-Demand Broadcasting

As described above, one of the many possible mechanisms for data dissemination uses on-demand (i.e., aperiodic pull) broadcast of data. In a typical scenario, two independent networks are used: a terrestrial network for sending pull requests to the server, and a "listen only" satellite downlink over which the server broadcasts data to all of the clients. When a client needs a data item (e.g., a web page, database object, map, etc.) that it cannot find locally, it sends a request for the item to the server. Client requests are queued up (if necessary) at the server upon arrival. The server repeatedly chooses an item from among these requests, broadcasts it over the satellite link, and removes the associated request(s) from the queue. Clients monitor the broadcast and receive the item(s) that they require.

2.1 Scheduling with RxW

In a large-scale implementation of such a system, an important consideration is the scheduling algorithm that the server uses to choose which request to service from its queue of waiting requests. We have developed a novel on-demand broadcast scheduling algorithm, called RxW [Akso98a], which is a practical, low-overhead and scalable approach that provides excellent performance across a range of scenarios.

The intuition behind the RxW scheduling algorithm is to provide balanced performance for hot (popular) and cold (not so popular) pages. This intuition is based on our observations of previously proposed algorithms, which failed because they favored one class of items over the other, or because they were too expensive to be used in a real system. The RxW algorithm schedules the page with the maximal $R \times W$ value where R is the number of outstanding requests for that page and W is the amount time that the oldest of those requests has been waiting for the page. Thus, RxW schedules a page either because has many outstanding requests or because there is at least one request that has waited for a long time.

The search algorithm is made efficient by using two sorted lists of requests (one ordered by R values and the other ordered by W values) threaded through the service queue. The algorithm prunes the search space by alternating between the two lists, each time, bounding the search on the other list. When the limit on

one of the lists is reached, the page with the maximal RxW has been found and can be broadcast. In our experiments, this pruning technique was found to reduce the size of the search space by over 70%.

We also developed an approximation-based version of the algorithm to provide even further reductions in scheduling time with only minimal impact on scheduling quality. By varying a single parameter, this algorithm can be tuned from having the same behavior as the *RxW* algorithm described previously, to being a *constant time* approach.

We implemented *RxW* and its approximate version on our dissemination toolkit. Experiments with the toolkit verified the effectiveness and efficiency of *RxW* and showed that for our particular configuration, the approximate version was able to significantly outperform the full version, providing fast scheduling while still producing a high-quality broadcast schedule.

2.2 Data Staging

While the *RxW* algorithm is a practical approach to on-demand broadcast scheduling, it like all previous work on broadcast scheduling, does not account for the need to obtain the items before they can be broadcast. In many large-scale applications data may not be available immediately when required by the scheduler. There are many applications that involve large amounts of data that cannot be cost-effectively Stored in main memory. Furthermore, in a wide-area distributed system such as the WWW, the data to be broadcast is likely to reside at a remote site. In either case, data items must be retrieved and brought into the server's main memory before they can be broadcast. The need to fetch data from various locations produces large variance in service times, which can destroy the performance of traditional scheduling heuristics. Thus, a communications-centric approach that ignores data management issues can result in significant degradation of broadcast efficiency. For this reason, we have investigated the coordination of broadcast scheduling with the management of the data items to be broadcast. We refer to this integrated functionality as *data staging.*

We have developed three complementary approaches to data staging All three approaches exploit the information on page popularity that is maintained by *RxW* and have been integrated with *RxW* in our dissemination testbed. The three data staging approaches are the following:

- *Opportunistic Scheduling:* In a large-scale broadcast system, broadcast bandwidth is the key shared resource, and thus, it is crucial utilize it to the fullest extent. It is a well-known property of broadcast scheduling that the optimal allocation of bandwidth to items is proportional to the ratio of the *square roots* of their access probability. The practical implication of this is that the broadcast effectiveness is not greatly effected by small deviations from its optimal allocation. We exploit this property by sometimes broadcasting sub-optimal, but memory resident data items, while the scheduled items are being brought into the server's cache. We have developed three alternative approaches for choosing these alternative items to be broadcast.

- *Caching:* An obvious way to reduce the need for fetching data items is to make the best use of the available memory space on the server. The key to successful caching is to retain those items that are most likely to be scheduled. The *RxW* algorithm is able to provide very good hints for differentiating between hot and cold items. We exploit this property to make intelligent caching decisions.

- *Prefetching:* Another method to reduce access latency is to predict which items will be broadcast in the near future and bring them into the cache before they are actually scheduled for broadcast. Since it is the responsibility of the caching policy to keep hot items available, prefetching focuses only on cold items, which are not likely to be in the cache. The *RxW* algorithm can help identify cold items that are likely to be broadcast in the near future.

Our performance experiments using both synthetic workloads and WWW server traces have shown that data staging concerns are indeed crucial, and that these approaches are effective (to varying extents) in providing substantial performance improvements for on-demand broadcast.

3 Self-Adaptive User Profiles for Large Scale Data Delivery

User profiles, which encode the data needs and interests of users, are lie at the heart of any dissemination-based information systems. ¿From the user's viewpoint, a profile provides a means of *passively* retrieving relevant information. A user can submit a profile to a push-based system once, and then continuously receive data that are relevant to him or her in a timely fashion without the need for submitting the same query over and over again. This automatic flow of relevant information helps the user keep pace with the ever-increasing rate of information generation. From the system point of view, profiles fulfill a role similar to that of queries in database or information retrieval systems. In fact, profiles are a form of continuously executing query. In a large data dissemination system, the storage and access of user profiles can be be resource-intensive. Additionally, given the fact that user interests are changing over time, the profiles must be updated accordingly to reflect up to date information needs.

We have developed an algorithm called *Multi-Modal* (MM), for incrementally constructing and maintaining user profiles for filtering text-based data items [Ceti99]. MM can be tuned to tradeoff effectiveness (i.e., accuracy of the filtered data items), and efficiency of profile management. The algorithm receives relevance feedback information from the users about the documents that they have seen (i.e., a binary indication of whether or not the document was considered useful), and uses this information to improve the current profile. One important aspect of MM is that it represents a user profile as multiple keyword vectors whose size and elements change dynamically based on user feedback.

In fact, it is this *multi-modal* representation of profiles which allows MM to tradeoff effectiveness and efficiency. More specifically, the algorithm can be tuned using a threshold parameter to produce profiles with different sizes. Let us consider the two boundary values of this threshold parameter to illustrate this tradeoff: When the threshold is set to 0, a user profile is represented by a single keyword vector, achieving an extremely low overhead for profile management, but seriously limiting the effectiveness of the profile. At the other extreme, if the threshold is set to 1, we achieve an extremely fine granularity user model, however the profile size equals the number of relevant documents observed by the user, making it impractical to store and maintain profiles. Therefore, it is more desirable to consider intermediate threshold values which will provide an optimal effectiveness/efficiency tradeoff for a given application.

We evaluated the utility of MM by experimentally investigating its ability to categorize pages from the WWW. In particular, we tested its ability to learn (human-generated) categories provided by the Yahoo! index. Our focus was on the tradeoffs between profile sizes and effectiveness (using non-interpolated average precision as our primary effectiveness metric). The evaluation demonstrated that MM can achieve significantly higher precision values with only a modest increase in profile sizes. Additionally, MM was able to achieve precision values with small profiles that were comparable to, or in some cases even better than those obtained with maximum-sized profiles. The details of the algorithm, experimental setting, and the results are discussed in [Ceti99].

4 Information Quality Metrics

A subproject of this work has investigated the connections between data dissemination policy and task utility. While there are various quality metrics that can be gathered directly on an information stream, those metrics might not translate directly into the "quality" with which an information-dependent task is performed.

One of the aims of the subproject was to gain insight on how the intuitive notion of the "importance" of an information stream can be translated into a lower-level system control policy. Some proposals that we saw put forward in the BADD implementation effort, such as interpreting importance as straight priority, seemed to have unpleasant consequences, such as task starvation.

To get better qualitative and quantitative handle on task utility, we concetrated on a particular category of task, namely those that depend strongly on a situation estimate. By "situation estimate", we mean a computer-based model of some part of the physical world. Examples of situation estimates might include

terrain elevation and cover, major lines of communication for vehicles, and location and patterns of emission of radio broadcast sites. By "stong dependence" of a task on a situation estimate, we mean a task whose performance directly correlates with the accuracy and currency of the estimate. For example, a task to detect when red-force vehicles cross a certain boundary could strongly depend on an estimate of vehicle tracks.

One result of our analysis of utility of information streams for such tasks is the recognition that utility cannot simply be assigned to individual information items in a stream. There are several reasons that utility can't be assigned to items in isolation. One is that the effect of losing one item can depend on whether other items in the information stream were received at the task site. Losing one report on a vehicle position might have minor effect if reports three seconds earlier and three seconds later were received. On the other hand, loss of one information item might render others useless, or of much diminished value. For example, if two listening stations are reporting the time, bearing and frequency of radio broadcasts, losing a report from one may make the corresponding report from the other useless for purposes of locating an emitter by triangulation.

In addition, the relative contribution of a given information item might depend on the latency with which it is received. A position report on a vehicle will lose value as it is delayed, especially after a subsequent report is recieved on the same vehicle.

Thus, schemes for managing information streams that are based on assigning static values to individual information items are likely to result in sub-optimal dissemination policies. Rather, such policies need to be evaluated by their effect on an information stream as a whole as it influences task performance.

To gain a better understanding of these issues, we have begun a simulation effort to evaluate different channel-scheduling and item-dropping schemes relative to task performance. As an initial simplifying assumption, we have approximated task performance by situation estimate accuracy. We essentially measure the integral of estimate error over time (so higher scores mean poorer task performance).

Our initial simulations are looking at different channel-scheduling policies relative to this measure. We have pure-priority, round-robin and weighted-deficit round-robin schedulers running. Initial simulations reveal that pure-priority scheduling benefits the task using the highest-priority information stream, to the detriment of all tasks at any lower priority. Round-robin ensures that all tasks get a share of channel capacity for their information streams, but fails to reflect relative importance of task. Weighted-deficit round-robin is a so-called "proportional share" scheduling algorithm, and allows more important streams to receive a larger share of resources, while not starving or arbitrarily delaying other streams.

Currently, different streams are served on a first-come, first-served basis, and queues of information items can grow without bound. Planned extensions will incorporate dropping policies and reordering of streams.

5 Channelization

In a publish-subscribe system, a profile describes a client's interest in a set of data items, and these profiles are used by a server (i.e., a data source) to send data to appropriate clients.

Often, the information that is sent to clients is broken up into a number of channels. A channel is a tranmission medium with fixed bandwidth over which data can be sent. Typically, clients can listen to a constrained set of channels. These channels can be physical manifestations requiring specialized tuners at the client end (e.g., satellite systems), or they can be virtual by mutiplexing multiple channels on a single physical channel. In the case of virtual channels, they provide a way of narrowing the focus of what a client must filter.

5.1 Simulator Structure

We have constructed two simulation models to study a satellite-based publish-subscribe system. Each model has the following fundamental characteristics. A data source produces updates to data based some update

distribution with exponential interarrival rates. Clients are each connected to some number of satellite channels, and a delivery mechanism uses client profiles to send data source updates on channels so as to satisfy all client interests. The delivery mechanism contains three pieces. The first is a Profiler which matches updates with profiles to determine whether there exists a client interested in the update. Next, is the Mapper that determines the channels on which to send updates based on the channels on which interested clients are connected. Last, is the the Scheduler which determines the order in which to send updates.

The first model looks at two problems. First, how are clients assigned to channels, and second, how are updates mapped to channels.

Client-to-channel assignment is a challenging problem because both client interests and data update rates have to be considered while attempting to make efficient use of bandwith. This goal is achieved by trying to group clients with similar profiles. The success of an algorithm is measured by the clients getting less of what they don't want, and the Mapper making fewer copies of updates.

Page-to-channel mapping is also an interesting problem. Based on a client-to-channel assignment, the Mapper determines on which channels to map updates. The mapping can be either static or dynamic. With a static mapping, the channels that a data item is mapped to is fixed. With dynamic mapping, the channels that a data item is mapped to is determined when the update is received by the Mapper. The goal of both types of algorithms is to minimize the number of channels an update is sent on, and also, to reduce the load accross all channels.

The second model reverses the approach of the first model. Data items are first mapped to channels, and based on this mapping clients select channels on which to connect.

Here, the mapper effectively accomplishes both the task of client-to-channel assignment and page-to-channel mapping with one algorithm. The mapper assigns data items to channels based on knowledge of profiles and data item update rates. It attempts to cluster profiles so as to minimize the load accross channels while also attempting to reduce the number of channels on which each client has to listen.

Client-to-channel assignment can be done in two ways: the data delivery mechanism can tell each client on which channels it should listen, or with a guide of page to channel mappings, each client can select appropriate channels.

5.2 Simulator Measurements

We use *average staleness* as our metric of how well we are doing. Average staleness is the difference between the time at which the client receives an update minus the time at which the update actually occured averaged over all clients that received that update. This represents an average delay for items to work their way through the system. An important realization is that once the flow into the system exceeds a given rate such that one or more of the queues becomes unstable, our average staleness measurements become meaningless. Thus, we use a *differential flow rate* (DFR) to determine when the system becomes unstable. It is the point at which the DFR becomes non-zero that is of interest. In other words, an algorithm that delays this point is doing better than one that does not.

In the first simulation model, we have studied various client-to-channel assignment algorithms. We have shown that the choice of an assignment mechanism here is related to the choice of a mapper algorithm at the server. We have also studied several server mapping algorithms. One study shows that if D is a measure of channel load and C is a measure of the number of copies that are made, an algorithm based on a combined metric such as DC outperforms one that considers only D or one that considers only C.

Both of these simulators are working quite well and provide a good testbed for further study of channelization algorithms. This includes algorithms for profile matching, channel assignment, channel mapping, and channel scheduling.

6 References

[Akso98a] D. Aksoy, M. Franklin "Scheduling for Large-Scale On-Demand Data Broadcasting" *IEEE INFOCOM '98*, San Francisco, March, 1998, pp 651-659.

[Akso98b] D. Aksoy, M. Altinel, R. Bose, U. Cetintemel, M. Franklin, J. Wang, S. Zdonik. "Research in Data Broadcast and Dissemination", (Invited Paper), *1st International Conference on Advanced Multimedia Content Processing*, Osaka University, Osaka, Japan, November, 1998, pp 196-210.

[Akso98c] D. Aksoy, M. Franklin "A Scheduling Approach for Large-Scale On-Demand Data Broadcast" *ACM/IEEE Transactions on Networking*, accepted pending revision, December, 1998.

[Akso99] D. Aksoy, M. Franklin, and S. Zdonik. "Data Staging for On-Demand Broadcast", *In Preparation*, June, 1999.

[Alti99] M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro and S. Zdonik. "DBIS-Toolkit: Adaptive Middleware for Large Scale Data Delivery", (Demonstration Description) *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, Philadelphia, PA, June, 1999, pp 544-546.

[Ceti99] U. Cetintemel, M. Franklin, and C. Giles, "Self-Adaptive User Profiles for Large Scale Data Delivery" *Submitted for Conference Publication*, June, 1999.

[Fran97] M. Franklin, S. Zdonik, "A Framework for Scalable Dissemination-Based Systems", *Proc. ACM Conf. on Object-Oriented Programming Systems, Languages, and Applications, (Invited Paper)*, Atlanta, GA, October, 1997, pp94-105.

[Fran98] M. Franklin, S. Zdonik. "Data in Your Face: Push Technology in Perspective", *Proc. ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 98)*, Seattle, WA, June, 1998, pp 516-519.

Appendix A

Scheduling for Large-Scale On-Demand Data Broadcast

Scheduling for Large-Scale On-Demand Data Broadcasting

Demet Aksoy, Michael Franklin
Computer Science Department and UMIACS
University of Maryland, College Park MD
demet@cs.umd.edu, franklin@cs.umd.edu

Abstract— Recent advances in telecommunications have enabled the deployment of broadcast-based wide-area information services that provide on-demand data access to very large client populations. In order to effectively utilize a broadcast medium for such a service, it is necessary to have efficient, on-line scheduling algorithms that can balance individual and overall performance, and can scale in terms of data set sizes, client populations, and broadcast bandwidth. In this study we introduce a parameterized algorithm that provides good performance across all of these criteria and can be tuned to emphasize either average or worst case waiting time. Unlike previous work on low overhead scheduling, the algorithm is not based on estimates of the access probabilities of items, but rather, it makes scheduling decisions based on the current queue state, allowing it to easily adapt to changes in the intensity and distribution of the workload. We examine the performance of the algorithm using a simulation model.

I. Introduction

A. On-demand Data Broadcast

Broadcast-based information systems are becoming increasingly popular due to advances in telecommunications, interconnectivity and mobile computing. Compared to traditional unicast data transfer, broadcasting can be much more efficient for disseminating information to large numbers of clients in applications where there is a high degree of commonality among client interests. With unicast, a data item must be transmitted at least once for every client who requests it, resulting in scalability problems as the client population increases. The World Wide Web has provided numerous examples of such situations, such as election result servers, sporting event kiosks, stock market tickers, etc. The access delays associated with such sites on the WWW during periods of heavy use demonstrate the inefficiencies of unicast delivery for dissemination-oriented applications.

The advantage of broadcast for data dissemination is that each transmission of an item can satisfy the needs of potentially many clients. Several forms of data broadcasting have already been introduced commercially. Intel has been broadcasting data along with normal TV signals [Intel94]. Hughes Network Systems is using satellites for delivering Internet content [DirecPC] and plans to incorporate broadcast technology. Broadcasting using cable technology is being developed by Hybrid Networks Inc. [Hybrid] and others. There has also been tremendous improvement in the bandwidth that is available for data broadcast. For example, the Teledesic system is expected to provide bandwidth of 155.52Mbps up to 1.244Gbps [Teledesic].

This work has been partially supported by the NSF under grant IRI-9501353, by Rome Labs agreement number F30602-97-2-0241 under DARPA order number F078, and by research grants from Intel and NEC.

B. Our Focus

While broadcast technology continues to advance in terms of both ubiquity and bandwidth, improvements in interconnectivity are fueling explosive growth in the amount of data available on-line and the number of clients who wish to access that data. In this paper, we focus on scheduling algorithms for *dissemination-oriented applications* in which a large and possibly changing client population requests data items from an information source equipped with a data broadcasting capability. The challenge in developing scheduling algorithms for such on-demand data broadcast is to provide scalable performance that balances average and individual (i.e., worst case) responsiveness using the shared broadcast medium. Such algorithms must cope with large databases, large client populations with dynamically changing interests and composition, and with high broadcast bandwidth.

As indicated by the preceding discussion, a scheduling approach for large-scale, on-demand data broadcast must balance different requirements. The algorithms that have been developed to date (e.g., [Dyke86], [Wong88], [Vaidya96], [Su97]) have failed to meet one or more of these needs. Some approaches have used simple scheduling policies such as FCFS (First Come First Served), which provide average case performance that is significantly lower than what could be supported by the broadcast medium. More sophisticated approaches aimed at providing better performance have been based on assumptions that limit their applicability, such as assuming very small database sizes, static data access probabilities (thereby limiting the ability to adapt to changing client needs), and/or ignoring the overheads associated with making intelligent scheduling decisions.

A key element that has been missing from the previous work is a comprehensive set of metrics for on-demand data broadcast in large-scale data dissemination environments. In this paper, therefore, we first outline the performance criteria that must be addressed by such scheduling algorithms. These criteria include: average and worst case performance, scheduling overhead, and robustness in the presence of certain environmental changes. We show how existing algorithms fail to meet one or more of these criteria.

We then define a parameterized algorithm, called RxW, that performs well for all of these metrics and furthermore, can be tuned to focus on scheduling overhead, average waiting time, or worst case wait time according to the needs of a particular application. RxW is robust to changes in the client population and workload because it makes scheduling decisions based only on the current queue state, rather than depending on estimates of data item access probabilities.

The remainder of the paper is structured as follows. In Section II we give a brief description of the problem and define the important criteria for evaluating scheduling algorithms for large-scale broadcast. We then describe how previously proposed algorithms measure up to these criteria. In Section III we develop several variants of a new algorithm, called *R*x*W*, which has low overhead and provides good average and worst case performance. Section IV presents an evaluation of the algorithm in terms of its performance, scalability, and robustness to workload changes. Section V discusses related work. Finally, Section VI presents our conclusions.

II. Background

A. Environment

In this section we present a simple satellite-based broadcast scenario to motivate the scheduling problem we are addressing. In this scenario (depicted in Figure 1) clients use two independent networks for communicating with the server: a terrestrial network for sending requests to the server, and a "listen only" satellite downlink to receive data from the server, similar to Hughes Network System's DirecPC architecture [DirecPC] and other satellite data services. When a client needs a data item (e.g., a web page or database object) that it cannot find locally, it sends a request for the item to the server. Client requests are queued up (if necessary) at the server upon arrival. The server repeatedly chooses an item from among these requests, broadcasts it over the satellite link, and removes the associated request(s) from the queue. Clients monitor the broadcast and receive the item(s) that they are waiting for.

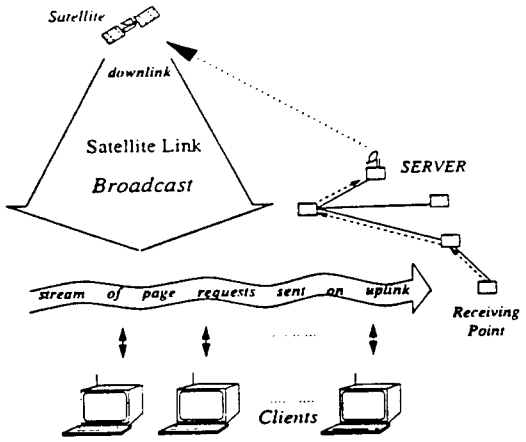


Fig. 1. Example Data Broadcasting Scenario

The focus of this paper is on the *scheduling* algorithm used by the server to choose the item to broadcast among those that have been requested. Because a single broadcast of an item satisfies *all* of the outstanding requests for the item, a good scheduling algorithm has the potential to greatly improve the effectiveness of the broadcast.

Similar to the previous work on broadcast scheduling we make the following assumptions about the environment. First, we assume that data items are fixed-length (e.g., database pages) so that the broadcast bandwidth can be divided into equal length, item-sized "slots". In the remainder of the paper, we refer to such fixed-sized items as "pages" and we refer to the length (in time) of a broadcast slot as a *broadcast tick* and use such ticks as the unit of time measure.[1] Second, we assume that clients continuously monitor the broadcast after they make a request and we do not consider the effects of transmission errors, so that all clients that are waiting for an item receive that item when it is broadcast by the server. Finally, we ignore the delay for sending requests via the client-to-server uplink, which we expect to be small compared to the latency of obtaining broadcast items from a moderately or heavily loaded server.

B. Performance Issues

Given the application environment described so far, we can now state our criteria for evaluating broadcast scheduling algorithms for large-scale data dissemination.

B.1 Responsiveness

The success of a scheduling algorithm is determined by its ability to get requested data to the clients quickly. In this regard, the first important metric, *average waiting time* is the amount of time on average, from the instant that a client request arrives at the server, to the time that the requested item is broadcast. Second, *worst case waiting time* is the maximum amount of time that any client request will have to wait in the service queue to be satisfied.

B.2 Scheduling Overhead

Because of the requirement for scalability, a key aspect of this study is the consideration of scheduling overhead at the server. Overhead is examined in two categories:

1. *Request Processing* - When a new request arrives at the server, the server must quickly decide whether or not to place an entry in the request queue for the requested item and/or update and possibly restructure the queue contents. The speed of such processing limits the rate at which requests can be processed by the server, effectively placing a limit on the scalability in terms of request arrival rate (e.g., number of clients supported).
2. *Scheduling Decisions* - On every broadcast tick the server must choose a page to broadcast. If the decision overhead is excessive the server may not be able to support the increased broadcast bandwidth or larger database sizes.

The two types of scheduling overhead are related: for example, doing extra work when requests arrive (e.g., keeping sorted lists of page access probabilities) can reduce the cost of making scheduling decisions at the expense of an increased cost for processing requests. The proper trade-off between these costs is dictated by the types of scalability that are important for a particular environment.

[1]The fixed length assumption simplifies the algorithm descriptions and analysis. Recent work in broadcast scheduling has shown how to extend scheduling algorithms to incorporate variable-length items [Vaidya96], [Su97].

B.3 Robustness

In order to achieve the goals of responsiveness and scalability, a scheduling algorithm will typically employ approximations and/or heuristics. Such heuristics must not be based on static information that will cause the algorithm to perform poorly if the workload or the environment changes.

C. Previous Algorithms

As stated in the Introduction, several algorithms for on-demand broadcast scheduling have been proposed previously. In this section, we describe existing algorithms and discuss their limitations with respect to the criteria that were outlined in the preceding section. Dykeman et al. [Dyke86] studied on-line scheduling algorithms, and were the first to point out that traditional FCFS scheduling would provide poor average wait time for a broadcast environment when the access distribution for data items was non-uniform. They proposed several algorithms aimed at providing improved performance. The algorithms studied in [Dyke86] (and later described in [Wong88]) are the following:

- **First Come First Served (FCFS)**: broadcasts the pages in the order they are requested.
- **Most Requests First (MRF)**: broadcasts the page with the maximum number of pending requests.
- **Most Requests First Lowest (MRFL)**: is essentially same as MRF, but breaks ties in favor of the page with the lowest access probability.
- **Longest Wait First (LWF)**: selects the page that has the largest total waiting time, i.e., the sum of the time that all pending requests for the item have been waiting.

Figure 2 plots the average waiting time (in broadcast ticks) for a workload with a database of 10000 pages. Client requests for pages are generated using a Zipf distribution of maximum skewness (parameter θ=1). The results were generated using the simulation environment and default parameters that we will describe in Section IV. As in [Dyke86], the overheads associated with running the scheduling algorithm at the server are not modeled here.

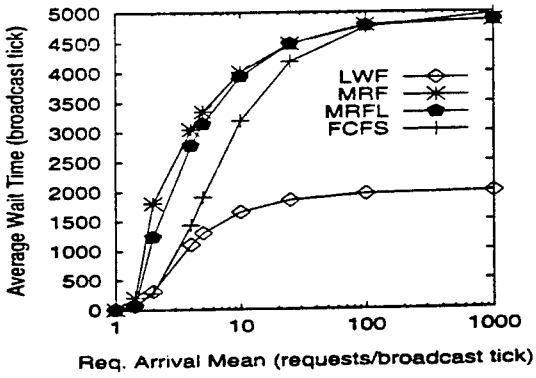


Fig. 2. Average Wait Time for Algorithms of Dykeman et al.

As can be seen in the figure, the best performance overall in this case is provided by LWF. As would be expected, the average wait time increases for all algorithms as more requests are introduced. Perhaps less predictably, however, the average response time eventually levels off and becomes insensitive to additional load. At this point, the remaining algorithms are approximately 2.5 times slower than LWF. Unfortunately, LWF is not a practical algorithm for a large system, as at each broadcast tick, it recalculates total accumulated wait time for *every* page with pending requests in order to decide which page to broadcast. For a high-bandwidth system with a large database, such a scheduling algorithm would likely become a bottleneck.[2] MRF and MRFL algorithms were introduced as lower-overhead alternatives.

The results of Figure 2 agree with those of [Dyke86], [Wong88] except for two key points. First, the earlier work did not investigate the performance of the algorithms under very high loads, so it did not identify the flattening of the performance curves for all of the algorithms under high load. Second, in the earlier study MRFL was seen to provide a performance between that of FCFS and LWF. Thus, MRFL was proposed as a lower-overhead replacement for LWF. In contrast, our results show that MRFL has poor performance relative to LWF and thus, is not a reasonable replacement. The differences in the conclusions stem from the fact that the earlier study was performed using a very small database (100 items, compared to 10000 in Figure 2). As the size of the database increases, the probability of having a tie for the largest number of requests diminishes. Without ties, MRFL degenerates to MRF, and has relatively poor performance.

The poor performance of MRFL for larger systems has also recently been shown by Su and Tassiulas [Su97]. In that paper, they propose an alternative algorithm, called PIP-0.5 (Priority Index Policy), that performs as well as LWF in average wait time. Unlike LWF, PIP-0.5 can be implemented with an O(1) complexity for choosing the next page to broadcast. PIP-0.5 falls short of our performance criteria, because it is based on estimates of the probability of access for each item. As a result, its usefulness is limited to fairly stable environments where those probabilities do not often change significantly. Furthermore, the history mechanism that must be employed to obtain such probability estimates can result in additional overhead, particularly for very large data sets. Algorithms based on access probabilities and broadcast histories have also been proposed by Vaidya and Hameed [Vaidya96]. These algorithms have similar performance to the PIP-0.5 algorithm, and also share that algorithm's limitations in terms of robustness to changing workloads.

III. *RxW*: A Parameterized Algorithm

We now describe a new broadcast scheduling algorithm, called *RxW*, which is a practical, low-overhead, scalable approach that provides excellent performance across a range of scenarios.

[2] In our implementation of LWF, we found that using one processor of a DEC Alpha 2100 4/275 server and assuming a broadcast bandwidth of 155.52 Mbps, the LWF algorithm became a bottleneck with a database size of 5543 8KByte pages.

A. Intuition

The results shown in Figure 2 demonstrated that MRF and FCFS have poor average case performance compared to the higher-overhead LWF algorithm. Probing more deeply into their performance in this case leads to a very important observation. Figures 3(a) and 3(b) show the performance of the three algorithms for the 10% most popular pages in the database (i.e., the "hot" pages), and the remaining 90% of the pages (i.e., the "cold" pages) respectively.[3]

As can be seen in the figures, MRF provides the lowest waiting time for hot pages, but its performance for cold pages is by far the worst of the three algorithms. In contrast, FCFS provides similar performance for both classes of pages leading it to have the worst performance for hot pages and the best for cold pages of the three. MRF chooses the page with the highest number of outstanding requests, so that requests for infrequently accessed pages must wait until sufficient requests have arrived. Since MRF is not a starvation-free algorithm; it is quite possible that a request for a very cold page is never satisfied. In contrast, FCFS is a fair algorithm in which the maximum time a request must wait is the same for all pages. This behavior causes it to spend more bandwidth on requests for cold pages. The fact that both algorithms favor one class of pages over the other results in their both having poor performance on average. In contrast, LWF provides good performance for both types of pages, resulting in better average performance overall. Based on these observations, we set out to combine the two low-overhead approaches (MRF and FCFS) in a way that would balance their strengths and weaknesses and provide a more even-handed treatment of hot and cold pages.

B. Scheduling with RxW

B.1 The Exhaustive Algorithm

We have developed a new scheduling algorithm called RxW (Requests times Wait), which provides good performance by combining the benefits of MRF and FCFS in a way that ensures scalability by having low overhead. RxW broadcasts a page either because it is very popular or because it has at least one long-outstanding request.

RxW maintains a service queue structure with a single entry for each page that has outstanding requests. Entries contain a page identifier (**PID**), the count of the number of outstanding requests (**REQcnt**), and a time-stamp of the earliest unsatisfied request for the item (**1stARV**). This structure is hashed on PID.

The exhaustive algorithm works as follows: *When a request arrives* at the server, the server performs a hash look up to find the entry of the requested page. If it finds an entry, then it simply increments the **REQcnt** for that entry. If no entry is found, then a new one is created with **REQcnt** initialized to 1, and **1stARV** initialized to the current time. *At each broadcast tick*, the server chooses to broadcast the page with the largest value of $(R * W)$ with $R =$ **REQcnt** and $W = clock - \mathbf{1stARV}$, where *clock* is the current time in broadcast ticks. The entry for this page is then removed from the structure. The exhaustive algorithm finds the page with the maximum value by simply performing a linear scan of *all* the entries. Note that the queue size is limited by N, the number of pages in the database. Thus, the exhaustive RxW algorithm is similar in overhead to other proposed O(N) algorithms, such as [Vaidya96]. Note however, that RxW makes decisions based only on the current state of outstanding requests, and does not depend on estimates of page access probabilities. A detailed analytical study of the average waiting time and the limiting behavior of RxW is provided in [Aksoy97].

[3] Recall that the popularity of the pages for this case were generated according to the Zipf distribution.

B.2 The Pruning Algorithm

As with other O(N) scheduling algorithms, the overhead of RxW scheduling can be reduced by performing more work during request processing in order to keep the request information better organized. In order to avoid searching the entire list of pages with outstanding requests, we thread two sorted lists through the request queue structure: The **Wait** list is simply a FCFS queue based on **1stARV** in ascending order; the second list is the **Requests** list, which is kept sorted in descending order by **REQcnt**.

The **Wait** queue is maintained by simply appending a request entry to it when a request arrives for a page with no outstanding requests and removing a page's entry when the page is broadcast. The **Requests** list is maintained each time a request is received. This maintenance involves moving the affected entry to the proper place in the sorted list. We introduce an additional structure to speed up the maintenance process and to guarantee scalability in request arrival rates. This structure, called the "REQcnt index" contains pointers to each cluster of **REQcnt** value, i.e. all pages that have the same number of pending requests. Values are added to and removed from this index as the request structure evolves. Using this index, request processing is an O(1) operation (note that for any one request, an entry moves exactly one cluster up in the **Requests** list).

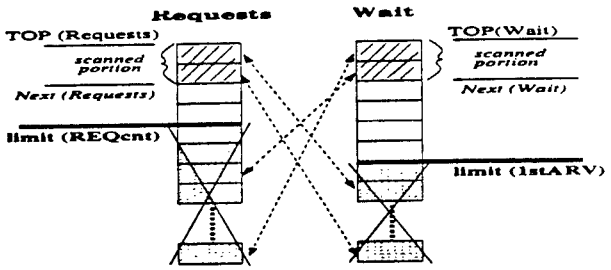
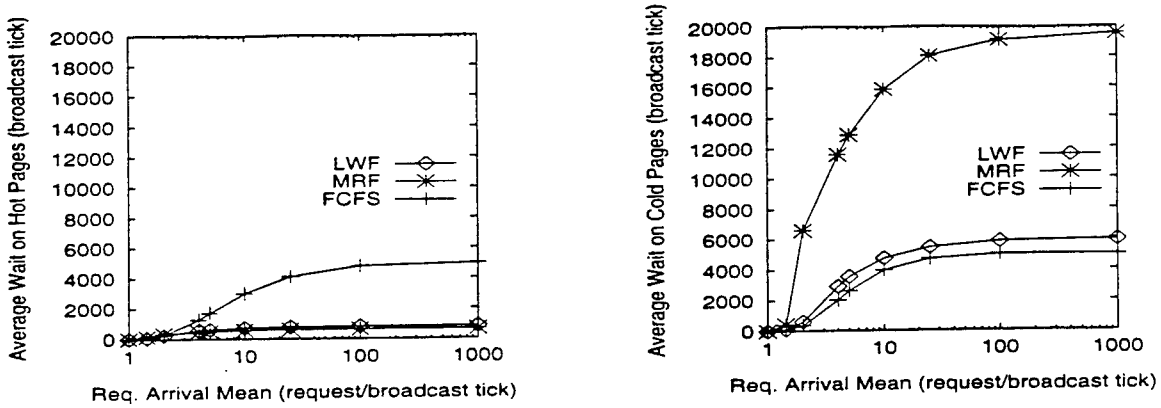


Fig. 4. Pruning the Search Space

The two sorted lists, (**Requests** and **Wait**), are used to prune parts of each list where there is no possibility of containing the entry with the highest RxW value. This pruning technique is depicted in Figure 4. Note that nodes connected by a dashed line belong to a single service queue entry. On the left is the **Requests** list, ordered by descending **REQcnt** and on the right is the **Wait** queue, ordered by ascending



(a) Hot Pages

(b) Cold Pages

Fig. 3. Average Waiting Time for Hot and Cold Page Requests

1stARV. When an entry is examined from the **Requests** queue, it is known that all entries not yet scanned have a **REQcnt** field that is less than or equal to that of the entry. Thus, if the recently examined entry has values $REQcnt'$ and $1stARV'$, the only way for a remaining entry to beat the current maximum RxW value seen so far (MAX), is if

$$clock - 1stARV' \geq \frac{MAX}{REQcnt'}$$

where *clock* is the current time. Thus, the entries that must be searched on the **Wait** queue are bounded by:

$$limit(1stARV') = min(1stARV', clock - \frac{MAX}{REQcnt'})$$

The same kind of pruning can be applied on the **Requests** list by scanning entries on the **Wait** queue. The pruning algorithm starts from the top of the **Request** list (thereby truncating the **Wait** queue) and then examining the top of the **Wait** queue (thereby truncating the **Requests** list) and alternating until the search reaches the bottom of one of the (truncated) lists. Note that the stopping condition is checked merely by comparing the limit values, rather than actually maintaining a pointer to the exact boundary on the list.

This mechanism prunes the search space while still guaranteeing that the search will return the page with the maximum RxW value. Thus, in the performance study that follows, we ignore the exhaustive algorithm and use the term "RxW algorithm" to refer to this pruning variant.

B.3 Approximation for Speed Up

Scheduling based on the highest RxW value makes efficient use of broadcast bandwidth. Such efficiency, however, comes at a price in terms of the overhead that is incurred at each scheduling decision. As stated in Section II-B, such overhead can ultimately limit scalability in terms of supportable bandwidth or database size. The pruning technique for RxW aims to reduce this overhead. As will be shown in Section IV-C, this technique is indeed effective — for example, in the main workload of our experiments this pruning resulted in a 72% savings in terms of the number of entries searched in order to find the maximum RxW-valued page. While such a substantial savings is helpful, it is probably not sufficient to keep the scheduling overhead from ultimately becoming a limiting factor as the system is scaled to the huge applications that will be enabled by the national and global broadcasting systems currently being deployed.

Based on insight gained from early experiments with the RxW algorithm, we realized that the scheduling overhead can be reduced dramatically by backing off from the requirement of maximality for the RxW value when making scheduling decisions. We have developed an *approximate*, parameterized variant of RxW that allows the search space to be reduced further, at the possible expense of making less efficient use of the broadcast bandwidth. By varying a single parameter, the algorithm can be tuned from having the same behavior of the RxW algorithm described so far, to a *constant time* approach that provides maximal scalability.

The parameterized version of RxW is based on two insights about RxW scheduling. First, we found that with highly skewed access patterns (as would be expected in many dissemination-oriented applications), the page with the maximum RxW value is typically found very near the top of at least one of the two lists (Requests or Wait). As a result, even the pruning-based RxW algorithm can spend substantial resources examining entries after it has already encountered the maximum-valued entry. The second insight is that given a static workload (i.e., in terms of request arrival rate and access probability distribution) the average RxW value of the page chosen for broadcast typically converges to some value. This latter insight is exploited to create a ***self-adapting*** approximation algorithm based on the RxW value of the most recently broadcast page. We take care, however, to ensure that the approximation works well even in the presence of a changing workload.

The approximation algorithm requires a single parameter called *accuracy*, which can be set to any value, 0 or greater.[4]

[4] Typically the accuracy parameter will be set to a value between 0 and 1. Larger values can be used, however. In the limit, setting accuracy

Scheduling works as follows: First, the algorithm maintains a self-adapting *threshold*, which is updated on every broadcast tick by averaging the current threshold value with the RxW value of the page that was chosen to be broadcast on that tick. To find the next page to broadcast the request structure is searched as in the regular (pruning) RxW algorithm, but rather than searching for the page with the *maximal* RxW value, the algorithm chooses the *first* page it encounters whose RxW value that is greater than or equal to *accuracy* × *threshold*. If no such page is found, then the algorithm acts like the regular RxW algorithm and returns the page with the maximum RxW value.

The setting of the *accuracy* parameter determines the performance tradeoffs between average waiting time, worst case waiting time, and scheduling overhead. The smaller the value of the parameter, the fewer entries are likely to be scanned. At an extreme value of 0, the algorithm simply compares the top entry from both the **Requests** list and the **Wait** queue and chooses the one with the highest RxW value. In this case, the complexity of making a scheduling decision is reduced to O(1), ensuring that broadcast scheduling will not become a bottleneck regardless of the broadcast bandwidth, database size, or workload intensity. In the following section, we examine the performance tradeoffs of several settings for the *accuracy* parameter. We refer to the approximation-based RxW algorithms as $RxW.\alpha$, where α equals the value of the *accuracy* parameter as a percentage (e.g., an α setting of 0.80 is called $RxW.80$).

IV. Experimental Results

A. Simulation Environment

Our experiments were performed using a simple simulation model of the system using CSIM [Schw86]. As with previous studies, the model is intended to capture only the quality of the schedule produced by the given scheduling algorithms. As such, it does not include the overheads of scheduling and request processing at the server. These costs have been described in the previous sections and are addressed in the scalability portion of the experiments (Section IV-C). Also in keeping with earlier studies, we do not model the costs of using the back-channel for sending requests from the clients to the server as these costs will be the same for all of the scheduling algorithms.[5] The broadcast channel is modeled as a server with a fixed rate of broadcast. We do not specify an absolute value for this rate, but rather, use *broadcast ticks* as our measure of time. This approach emphasizes that the results are not limited to any particular bandwidth and/or data item size, but rather, that they describe fundamental tradeoffs among the algorithms.

In the model, the client population is represented by a single request stream. The client population model generates non-blocking requests with exponential inter-arrival times with mean λ. We use an open system model since our work is aimed at supporting extremely large, highly dynamic client populations, and such client populations cannot be realistically modeled with a closed simulation system. The request pattern is shaped with a Zipf distribution [Knuth81]. This is a frequently used distribution for non-uniform data access. It produces access patterns that become increasingly skewed as its θ parameter increases from 0 (uniform access probability) to 1 (highly skewed). The *offset* and *freq* parameters are used to simulate interest shifts of the client population and the frequency of such shifts. The parameters and their settings are summarized in Table I.

B. Responsiveness

In the first experiment, we examine the responsiveness of several variants of the RxW algorithm, and compare them to the LWF and FCFS algorithms of Dykeman et al. As stated previously, the results we present here are measures of the quality of the scheduling choices made by the various algorithms, and do not take into account the overhead of scheduling and request processing. Under such assumptions, LWF (and similar algorithms such as PIP-0.5) have provided the best average case performance in previous studies. We report results for four variants of RxW: The pruning algorithm and the approximate algorithm with α values of 0.90, 0.80, and 0 (referred to as $RxW.90$, $RxW.80$, *and* $RxW.0$, respectively). Recall that $RxW.0$ examines only the top entry of each the two sorted lists of requests, and thus, makes scheduling decisions in constant time. The four variants of RxW allow us to investigate the tradeoffs between the exact and approximate approach, among the various α settings.

In Figure 5(a), we plot the *average* waiting time for each scheduling algorithm, as the mean request arrival rate is varied from 1 per tick to 1000 per tick along the x-axis (shown with a log scale). All algorithms exhibit similar performance here, with the average wait time increasing but ultimately leveling off as the request arrival rate is increased. This leveling off is a characteristic of broadcast data delivery to clients with shared interests and differs dramatically from what would be expected in a unicast environment.

Comparing the six algorithms, it can be seen that LWF and the RxW algorithm provide the best average performance (RxW even does slightly better for loads between 5 and 50 requests/tick). The good performance of RxW in this case demonstrates that the scheduling decision metric used by RxW is a reasonable substitute for that of LWF (the total waiting time), and even for perfect knowledge of access probabilities (as used by PIP-0.5, which also has similar performance to LWF). By far the slowest average performance in this case is provided by FCFS. As described in Section III-A, for a sufficiently loaded system FCFS allocates the same bandwidth to all accessed pages, regardless of their popularity, resulting in poor utilization of the broadcast.

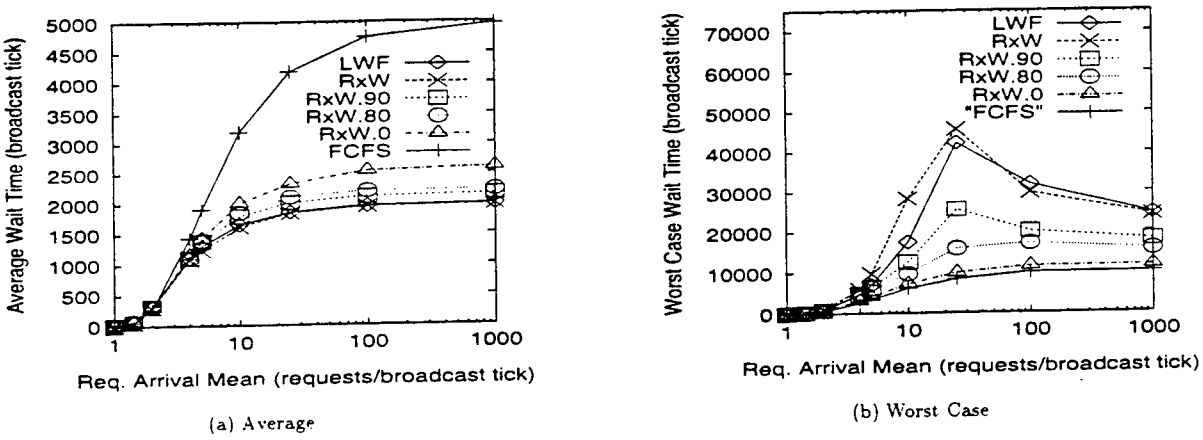
The results for the approximate RxW algorithm show that as the α parameter, which sets the accuracy for the broadcast schedule, is decreased the average wait time increases. For all three of the values shown, however, the performance

to ∞ results in the approximate algorithm behaving identically to the regular pruning-based RxW algorithm.

[5] The cost of back-channel requests becomes more important when trading off between server push and client pull over the broadcast as in [Acha97].

Symbol	Description	Default	Range	Unit
dbSize	Database Size	10000	[100-50000]	pages
λ	Mean Req Arv (exp)	100	[1-1000]	requests/tick
θ	Request Skew (zipf)	1.0	[0.0-1.0]	-
α	Approximation Accuracy	-	0, 0.80, 0.90	-
offset	Shift in Interest	0	[0-5000]	number of pages
freq	Interest Shift Freq	n/a	[1-100000]	/ 100000 broadcast ticks

TABLE I
SIMULATION MODEL PARAMETERS



(a) Average

(b) Worst Case

Fig. 5. Responsiveness Measures

is significantly better than that of FCFS. The approximate algorithm with an α value of 0.90 remains less than 10% slower than the maximal algorithm. Even the constant-time $RxW.0$ pays less than a 33% penalty compared to the maximal algorithm in the most extreme case here.

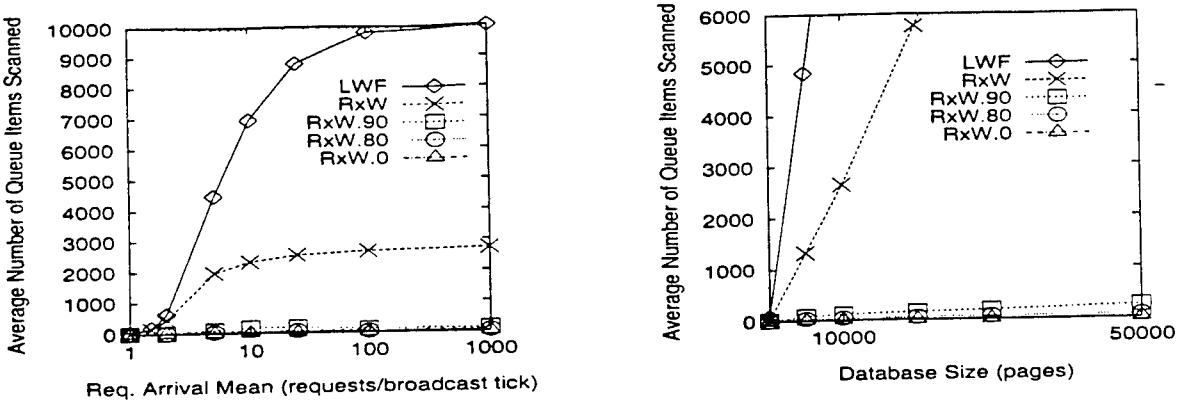
Figure 5(b) shows the *worst case* waiting time measured for the same experiment as Figure 5(a). That is, we plot the longest measured wait for *any* request that occurs during the simulation run. Note that the simulation was run one million broadcast ticks, so each page was broadcast at least several times. Although worst case performance has not been addressed by previous studies, it is an important metric for many applications. As can be seen in the figure, the ordering of the algorithms for worst case behavior is inverted compared to the average case. FCFS has the shortest worst case waiting time. In fact, once a page has been requested, it is guaranteed to be scheduled for broadcast before any other page is broadcast twice. Thus, its worst case behavior is bounded by the number of pages (10,000 in this case). In contrast, the LWF and RxW algorithms make no such guarantees — popular pages may be broadcast multiple times while requests for a less popular page are waiting.

For the approximate RxW algorithm, α serves as a knob for adjusting the worst case waiting time in the opposite direction of how it works for the average case. As α is decreased, the FCFS queue begins to play a larger role in the scheduling process, and thus, the behavior begins to look more like FCFS. As a result, for $\alpha = 0$, the worst-case behavior is within 15% of that of FCFS. Comparing Figures 5(a) and (b), it is apparent that the α parameter provides a flexible mechanism for trading-off worst case and average case waiting times for a particular application environment, and that it can also be set to balance both concerns reasonably well (e.g., $\alpha = 0.80$ in this case). In the next section we show that α can also be used to adjust the overhead of the scheduling decision process in order improve scalability.

C. Scheduling Overhead

As described previously, a critical aspect of scheduling algorithms for large-scale data broadcasting is scalability. The previous section focused on the performance of the algorithms in an ideal setting where there was no overhead for making scheduling decisions or processing requests. In practice, however, such concerns can limit the ability for on-demand systems to support large applications. As described in Sections II-C and III, all of the algorithms are fairly efficient in terms of request processing. They differ significantly, however, in terms of scheduling overhead. In this section we examine the question of scheduling overhead in more detail.

Figures 6(a) and (b) show the average number of request queue entries searched each time a scheduling decision is to be made (i.e., on each broadcast tick), as the request arrival rate and the database size are increased, respectively. Figure 6(a) corresponds to the same settings as the previous two graphs; that is, the database size is fixed at 10,000 pages and the request arrival rate is varied from 1 to 1000 requests/tick. As can be seen in the figure, LWF is by far the most expensive of the algorithms shown for making scheduling decisions, followed by the maximal RxW. Also, it can be seen that the



(a) Varying Request Rate (dbSize=10,000 pages)

(b) Varying Database Size (100 requests/tick)

Fig. 6. Scheduling Overhead Measures

approximate version of RxW provides tremendous savings.[6]

LWF is an exhaustive algorithm, and under high loads, there is at least one pending request for each data page. Thus, the scheduling cost of LWF is proportional to the number of distinct pages that are accessed by the client population, namely 10,000 pages in this case. For a fast broadcast bandwidth and/or large database size, the scheduling overhead of LWF could easily become a bottleneck. The maximal RxW algorithm, on the other hand, examines significantly fewer queue entries for each scheduling decision. In this case, it examines 2729 entries on average at a load of 1000 requests/tick; a savings of about 72.7%. It is important to note that as was shown in Figures 5(a) and (b), these savings in search complexity come at no cost in broadcast efficiency.

The savings provided by RxW's pruning algorithm, however, are dwarfed by the tremendous savings provided by the approximate version of the algorithm. In figure 5(a) at a load of 1000 requests/tick, $RxW.90$ and $RxW.80$ examine 116 and 39 entries respectively, for savings of more than 98.8% and 99.6% respectively. With α set to 0, the approximate algorithm scans only two entries providing maximum scalability in terms of search overhead.

Figure 6(b) shows even more striking results for the same algorithms, when the request rate is fixed at 100 requests/tick and the database is scaled from 1 to 50,000 pages. The overhead of LWF grows linearly with the database size, approaching the limit of one entry per page. The overhead of maximal RxW also grows linearly, but at a much slower rate. Finally, similarly to part (a) of the figure, the overhead of the approximate algorithms grows much more slowly, with $RxW.0$ remaining constant. The practical impact of these results is that the approximate RxW algorithm provides tremendous scalability in terms of request arrival rate and database size. In addition, although not shown here directly, these results indicate that RxW allows a broadcast system to scale in terms of the supportable broadcast bandwidth. Faster broadcast means shorter ticks, and thus, less time to make scheduling decisions. RxW is clearly capable of making fast scheduling decisions across a large range of system sizes and workload intensities.

[6] Note that FCFS is not shown on these graphs. It is a O(1) scheduling algorithm and so is insensitive to the parameters varied here.

D. Robustness

RxW and its approximations do not depend on any long term measurements or estimates of data access probability distributions, which enables them to easily adapt to changes in workload. The approximations, however, do use a threshold value that is dependent on previous performance. To test the robustness of the RxW variants, we performed a detailed sensitivity analysis. Due to space constraints, we briefly summarize the results of that analysis here. The results are reported in more detail in [Aksoy97].

In one set of experiments, the skewness of the access pattern (i.e., θ) was varied between 1 (default) and 0 (a uniform distribution). As the skew is reduced, all of the algorithms converge to the same average waiting time (at $\theta = 0$). The relative ordering of the algorithms studied remained constant. A second set of experiments kept θ at 1 but varied the focus of interest (i.e., the most popular items) to different parts of the database. The results showed that the approximate variants of RxW were slightly less robust than LWF for infrequent shifts of interest but were significantly more robust than LWF as the frequency of interest shifts was increased. Finally, a third set of experiments introduced a sudden 20-fold spike or a 20-fold decrease in the arrival rate of client requests. The results showed that the number of request queue entries scanned by RxW and its approximations still remained far below that of LWF.

V. Related Work

In this paper we have presented a new on-demand scheduling algorithm for large-scale data broadcast. The directly relevant previous work on scheduling algorithms [Dyke86], [Wong88], [Vaidya96], [Su97] has been addressed in detail in Section II. In addition to this directly related work, there has been much recent interest in other areas of data broadcasting. A taxonomy of data delivery mechanisms (including various

forms of broadcast) along with a framework for describing dissemination-based systems is provided in [Fran97]. Some recent applications of dissemination-based systems include information dissemination on the Internet [Yan96], [Best96], Advanced Traveler Information Systems [Shekhar96] and dissemination using satellite networks [Dao96].

The Datacycle Project [Herm87], [Bowen92] at Bellcore investigated the use of a repetitive broadcast medium for database storage and query processing. An other early effort in information broadcasting, the Boston Community Information System (BCIS) is described in [Giff90]. BCIS broadcast news articles and information over an FM channel to clients with personal computers specially equipped with radio receivers. Recently, scheduling techniques from the real-time community have been applied to data broadcast by Baruah and Bestavros [Baru96]. The Broadcast Disks project [Acha95b] has investigated a number of aspects of data broadcast using periodic push including scheduling and client caching [Acha95a], prefetching [Acha96] and integrating push and pull over a broadcast channel [Acha97]. The issue of combining broadcast push and unicast pull is addressed in [Stath97]. The mobility group at Rutgers [Imie94] has done significant work on data broadcasting in mobile environments. A main focus has been on indexing in order to reduce power consumption at the mobile clients. Viswanathan [Vis94] has studied integrating push and pull for a mobile broadcast environment.

VI. Conclusions

In this paper we focused on the challenges of large-scale on-demand data broadcast introduced by high bandwidth broadcasting media such as satellite or cable networks. Unlike previous work, we began by proposing a comprehensive set of performance criteria for scheduling algorithms. These criteria include worst case as well as average response time, three types of scalability, and robustness to changes in the nature and or intensity of the workload. We then described how previous algorithms fail in one or more of these criteria.

We proposed a scheduling algorithm called RxW, that provides a balanced treatment of hot and cold pages in order to achieve a good overall performance. The algorithm uses a novel pruning technique to reduce the search space for making broadcast decisions. While the pruning was shown to be effective, it was observed that such an algorithm could still eventually become a bottleneck for very large applications.

We developed an *approximate*, parameterized variant of RxW that allows the search space to be reduced further, at the possible expense of making less efficient use of the broadcast bandwidth. By varying a single parameter, the algorithm can be tuned from the regular RxW algorithm, to a *constant time* approach that provides maximal scalability. We demonstrated the performance, scalability, and robustness of the different RxW variants through an extensive set of performance experiments.

In terms of future work, we plan to integrate the on-demand scheduling described here, with push-based and other forms of data delivery as part of a larger Dissemination-Based Information Systems (DBIS) framework as described in [Fran97]. We also plan to investigate the scheduling of broadcast for hierarchical broadcast environments.

Acknowledgements

The authors would like to thank to Ugur Cetintemel, Bjorn Thor Jonsson and Mustafa Uysal for their helpful comments on an earlier draft of this paper.

References

[Acha95a] S. Acharya, R.Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", Proc. ACM SIGMOD Conf., San Jose, CA, 1995.

[Acha95b] S. Acharya, M. Franklin, S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks", IEEE Personal Communications, 2(6), 1995.

[Acha96] S. Acharya, M. Franklin, S. Zdonik, "Prefetching from a Broadcast Disk", Proceedings of the International Conference on Data Engineering, New Orleans, LA, Feb 1996.

[Acha97] S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Data Broadcast", Proc. ACM SIGMOD, May 1997.

[Aksoy97] D.Aksoy, M.Franklin, "On-Demand Broadcast Scheduling", Technical Report, CS-TR-3854, University of Maryland, 1997.

[Baru96] S.Baruah, A.Bestavros, "Pinwheel Scheduling for Fault-tolerant Broadcast Disks in Real-time Database Systems", Technical Report TR-96-023, Boston University, August 1996.

[Best96] A.Bestavros,,C.Cunha, "Server-initiated Document Dissemination for the WWW", IEEE Data Engineering Bulletin, 1996

[Bowen92] T. Bowen, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture", CACM Vol 32, No 12, December 1992.

[Dao96] S. Dao and B. Perry, "Information Dissemination in Hybrid Satellite/Terrestrial Networks", Data Engineering, 19(13), 1996.

[DirecPC] Hughes Network Systems, *http://www.direcpc.com*, July 97.

[Dyke86] H.D. Dykeman, M. Ammar, J.W. Wong, "Scheduling Algorithms for Videotex Systems Under Broadcast Delivery", IEEE International Conference on Communications, Toronto, Canada, 1986.

[Fran97] M. Franklin, S. Zdonik, "A Framework for Scalable Dissemination-Based Systems", Proc. ACM OOPSLA Conf., 1997.

[Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communication", CACM, 37(10), 1994.

[Herm87] G.Herman, G.Gopal K.Lee, A.Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", Proc. ACM SIGMOD Conf., San Francisco, CA, May 1987.

[Hybrid] High-Speed Data Access for Teleworkers, *http://www.hybrid.com*.

[Imie94] T. Imeilenski, B.R. Badrinath, "Energy Efficient Indexing On Air", Proc. ACM SIGMOD Conf., Minneapolis, MN, May 1994.

[Intel94] Intel Corporation. Introduction to Intercast Technology, *http://www.intercast.com*, 1994.

[Knuth81] D. Knuth, "The art of Computer Programming - Volume II", Addison-Wesley, 1981.

[Schw86] H.D. Schewtman, "CSIM: A C-based Process Oriented Simulation Language", Proc. of the Winter Simulation Conf., 1986.

[Shekhar96] S. Shekhar, A. Fetterer and D.R. Liu, "Genesis: An Approach to Data Dissemination in Advanced Traveler Information Sytems", Data Engineering, 19(3), 1996.

[Stath97] K. Stathatos, N. Roussopoulos, and J.S. Baras, Adaptive Data Broadcast in Hybrid Networks, in Proc. VLDB, 1997.

[Su97] C.J. Su, L.Tassiulas. "Broadcast Scheduling for Information Distribution", Proc. IEEE INFOCOM, 1997.

[Teledesic] "Application of Teledesic Corporation for a Low Earth Satellite System in the Domestic and International Fixed Satellite Services," filed by Teledesic Corporation with the Federal Communications Commission, March 21, 1994.

[Vaidya96] N.H. Vaidya and S. Hameed. "Data Broadcast in Asymmetric Wireless Environments". Proc. of Workshop on Satellite-based Information Services (WOSBIS), New York, November, 1996.

[Vis94] S.R.Viswanathan, "Publishing in Wireless and Wireline Environments", PhD Thesis, Rutgers University, 1994.

[Wong88] J.W. Wong, "Broadcast Delivery", in Proceedings of IEEE,pp. 1566-1577, Dec. 1988.

[Yan96] T. Yan and H. Garcia-Molina, "Efficient Dissemination of Information on the Internet", Data Engineering, 19(13), 1996.

Appendix B

Data Staging for On-Demand Broadcast

Data Staging for On-Demand Broadcast

Demet Aksoy
University of Maryland
demet@cs.umd.edu

Michael J. Franklin
University of Maryland
franklin@cs.umd.edu

Stan Zdonik
Brown University
sbz@cs.brown.edu

Abstract—**Advances in broadcast technology and deployment, along with scalability concerns have made wide-area data broadcasting an increasingly promising data delivery alternative for large client populations. As a result, there has been significant effort towards developing on-line scheduling algorithms for data broadcast servers. To date, such scheduling algorithms have been aimed at optimizing broadcast bandwidth allocation, and have been based on the assumption that all data items are readily available in the server's main memory. This approach ignores the data management issues that arise when data items need to be fetched from secondary storage or from remote sites before they can be broadcast. Such *data staging* concerns, if ignored, can result in significant degradation of the broadcast efficiency. In this paper we propose three data staging solutions: opportunistic scheduling, server caching, and prefetching, that are closely integrated with the RxW broadcast scheduling algorithm [AF98]. We then use a data broadcasting testbed based on IP-Multicast to examine the performance of these various solutions. Our results show that data staging concerns are indeed crucial, and that the hints provided by the RxW scheduling algorithm can be effectively used to dramatically enhance the performance of a large-scale on-demand broadcast system.**

I. Introduction

A. Asymmetric On-Demand Broadcast

Advances in telecommunications enable new asymmetric infrastructures for high speed data transmission rates using satellite networks or cable television networks [Dir96], [Web99], [Cyb99], [Hom98], [Cha98]. The asymmetry is in the relative capacity of the dedicated downlink (from server to clients) and the dedicated uplink (from server to clients). Typically, the downlink bandwidth is much higher than that of uplink so as to better match the high response versus request data flow rate, e.g., a mouse click of a URL request versus the whole contents of the Web page. High-bandwidth links are becoming available both on terrestrial cable networks and satellite networks. For example, the Teledesic system is expected to provide bandwidths of 155.52Mbps up to 1.244Gbps [Tel94]. Cable technology offers bandwidths of 36Mbps per channel for downstream, with 110 channels or more [Dat99].

Currently most infrastructures are based on unicast data delivery method even though the network inherently provides broadcast capabilities. With unicast delivery a data item must be transmitted individually to each client that requests it. This results in serious scalability problems with increases in the client population. The client population meanwhile is showing an enormous growth with the improvements of interconnectivity. For instance, in 1998, IntelliQuest [Int98] has reported 62 million Internet users only in US. When we consider such large client populations, the high downlink bandwidth is not sufficient as a solution with unicast data transfer. In contrast to unicast, broadcast-based delivery allows a single transmission of an item to satisfy all clients that require that item. Compared to traditional unicast data transfer, broadcast can, therefore, be much more efficient for disseminating information to large client populations, especially for applications where there exists a high degree of commonality among client interests. It should be noted that broadcast data transfer on these emerging infrastructures is always as good as or better than unicast data delivery, since the downlink channel is a shared resource and can only employ one transmission at a time, i.e., can not parallelize the transmission on multiple connections as for point-to-point infrastructures. In this

This work has been partially supported by the NSF under grant IRI-9501353, by Rome Labs agreement number F30602-97-2-0241 under DARPA order number F078, and by research grants from Intel and NEC.

paper we focus on broadcast-based data dissemination for on-demand data service using the emerging infrastructures.

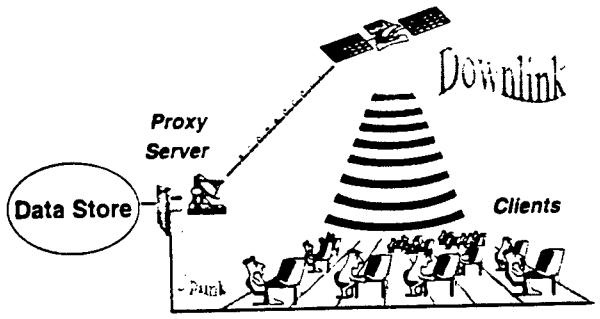


Fig. 1. Example Data Broadcasting Scenario

An example large-scale on-demand data broadcast environment is shown in Figure 1. In this scenario, clients send requests for data items to a proxy server via an independent uplink channel. The server receives and aggregates those requests in a service queue. Based on the received requests the server chooses data items to broadcast, and sends them to the clients over the shared downlink. Clients monitor the broadcast to receive the items they are interested in. This example depicts an environment similar to what could be provided using a Direct Broadcast Satellite infrastructure such as Hughes Network System's DirecPC [Dir96]. In this case, the uplink is a terrestrial, wired network while the downlink is a high-bandwidth satellite link. Other technologies are of course, also possible. For example, cable technology is also being used for data broadcasting [Hyb], and Internet multicast technology is improving [mee92].

B. Broadcast Scheduling and Data Staging

A key design consideration in the development of an on-demand data broadcast system is the *scheduling algorithm* used by the server. Such an algorithm aims to choose at each instance, the most beneficial data item to be broadcast based on the unfulfilled requests that have been received from clients. There has been significant work on the development of on-line scheduling algorithms (e.g., [DAW86], [Won88], [VH96], [ST97b], [AF98]). One main objective of the more recent studies has been developing efficient algorithms with low overhead so that the available broadcast bandwidth can be effectively utilized. To date, however, this work has been based on the assumption that all data items are readily available in the server's main memory to be broadcast [1] and has largely been focused on optimizing broadcast bandwidth allocation with the scheduling decisions made.

In many practical applications, however, data may not be available immediately when required by the scheduler. There are many applications that involve large amounts of data that cannot be cost-effectively stored in main memory. Furthermore, in a wide-area distributed system such as the WWW, the items to be broadcast are likely to reside at a remote site. In either case, data items must be retrieved and brought into the server's main memory before they can be broadcast. The need to fetch data from various locations produces large variance in service times, which can destroy the performance of traditional scheduling heuristics and can result in significant degradation of broadcast efficiency. For this reason, we have investigated the coordination of broadcast scheduling with the management of the data items to be broadcast. We refer to the process of making data items available for broadcast as *data staging*.

C. Data Staging Solutions

In this paper, we propose and investigate three complementary approaches to data staging. All three approaches are based on a broadcast scheduling algorithm called *RxW*, which we have previously shown to be efficient, effective, and robust for a wide range of workload characteristics [AF98], [AF99]. Intuitively, *RxW* broadcasts an item either if there are many outstanding requests for that item, or if there is at least one long-outstanding request for that item. *RxW* is described in more detail in Section III. The three data staging approaches we investigate are the following:

- *Opportunistic Scheduling:* It is crucial to keep the broadcast busy in order to fully exploit the available downlink bandwidth. We sometimes broadcast sub-optimal, but memory resi-

[1] The only exception of which we are aware is an early study by Dykeman et al. [DW88]. As discussed in Section VIII, this work is based on a scheduling algorithm that is not suitable for large systems, and includes solutions that require fine-grained control over the location of data on magnetic disks.

dent data items, when the optimal page to broadcast is being brought to server's memory. We investigate how to select such sub-optimal pages with only small deviations from the optimal allocation in the most efficient way.

- *Caching:* One obvious way to reduce the need for fetching data items is to make the best use of the available memory space on the server. The key to successful caching for on-demand broadcast servers is to retain those items that are most likely to be scheduled. The *RxW* algorithm can provide very good hints for identifying such pages because it differentiates between popular and not-so-popular items. We exploit this property to make intelligent caching decisions.
- *Prefetching:* Another method to reduce access latency is to predict which items will be broadcast in the near future and to bring them into the cache before they are actually scheduled for broadcast. We examine prefetching in an integrated caching/prefetching environment and exploit hints provided by *RxW* to identify items that are not cached but are likely to be broadcast in the near future.

We have implemented a data broadcast testbed using IP-Multicast on a cluster of Pentium-based workstations running Windows NT. We use this prototype to study the performance characteristics of the three data staging approaches we propose. In our experiments, we concentrate mainly on disk-resident data sets. The data staging approaches, however, are equally applicable to data residing at remote sites. In order to address this issue we also examine the effectiveness of the approaches as the latency for obtaining data is increased.

The remainder of this paper is organized as follows. In Section II we give a brief overview of the mechanism applied for an integrated scheduling and data staging solution. In Section III we briefly present the *RxW* scheduling algorithm. This is followed by the description of the prototype system in Section IV and experimental environment used to evaluate our data staging techniques. Sections V, VI, and VII present the data staging solutions that we propose and analyze their performance. Section VIII discusses related work. Section IX presents our conclusions and plans for future research.

II. Overview of the Mechanism

In this section we briefly explain the way the three data staging solutions interact with each other and the role of the scheduling algorithm during the process. As stated in the Introduction, previous work on on-demand broadcast scheduling did not address the issue of data staging. If data staging is completely ignored, the server would simply apply the scheduling algorithm and block on a cache miss (when the scheduled page is not in memory), waiting for the scheduled page to be faulted into memory. Obviously, such blocking would cause a significant degradation of broadcast bandwidth utilization, resulting in poor system performance. Instead, the server should initiate an asynchronous request to fetch the missed page. We will show that the implications of such asynchronous requests are not as straight forward as in the case of file systems.

We first explain the server mechanism where each data staging solution is designed to complement each other.

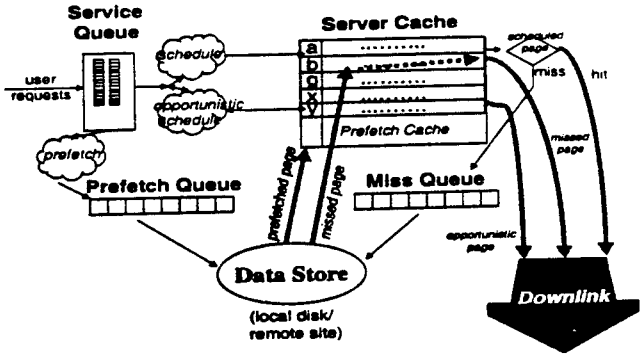


Fig. 2. Mechanism

The integrated mechanism is summarized in Figure 2. The user requests are queued at the server using a single entry per page. Since once a page is broadcast all requests on that page will be satisfied, we do not need to keep track of multiple entries for the same page in the service queue. All information used by the scheduling algorithm is incorporated into this single entry and is updated for each additional request made for the same page. In Figure 2 the server continuously selects a page to broadcast from the service queue. Initially the original scheduling algorithm is used, referred to as *schedule* in the figure. In this mode, if the scheduled page is in the cache (hit),

it is immediately broadcast. If the selected page is not in the cache (miss), an asynchronous request is initiated to fetch the page. This request is queued among previously requested pages, as shown in the *miss queue*. Then the page's entry is removed from the service queue. Later, when a missed page arrives in memory, it is broadcast as soon as the broadcast channel is available. Meanwhile, as the fetch takes place, the scheduler is run again for the next page to broadcast and the original scheduling process is repeated [2].

We place a limit on the number of outstanding I/O requests in the *miss queue* in order to avoid I/O thread thrashing. When this limit is exceeded, we change the mode of scheduling, and apply the opportunistic scheduling process, marked as *opportunistic schedule*. In this mode the server selects only cache-resident pages for broadcast [3]. That is, if the original scheduling was to select a non-cache-resident page, the opportunistic scheduler selects an alternative page that can be broadcast because it is available. Such pages that are selected not according to the original scheduling algorithm, but according to availability is shown as *opportunistic pages* in Figure 2. A key question for opportunistic scheduling is which cache-resident pages to select for broadcast during opportunistic scheduling mode. This issue will be examined in detail in Section V.

Another question that we address in this paper is the caching policy used by the server. That is, when a missed page arrives at the memory how we manage the server cache, so that the need to apply opportunistic scheduling is reduced. In Section VI we will analyze the server caching policy that we particularly propose for the RxW scheduling algorithm.

The final front of our attack to data staging problem is to bring the pages from the request queue to the cache before they are actually scheduled. This process is shown with *prefetch* in Figure 2. The pages that are selected for prefetch are queued in a *prefetch queue* and, when they arrive, are placed in a separate part of the cache, *prefetch cache*. Later, when they are scheduled the page is moved to the normal cache space accordingly. If a page that is being prefetched is scheduled before it arrives at server's memory, the page is marked to be treated as a *missed page* so that it will be placed in the normal cache space and will be broadcast as soon as it arrives. The decision on which pages to prefetch is examined in Section VII.

Different combinations of data staging solutions can yield interesting results for a broadcast environment as we will observe in Section V-B. In this paper, we will apply an incremental design in the order of opportunistic scheduling, server caching and server prefetching. All three approaches are closely related to the scheduling algorithm used.

III. The RxW Scheduling Algorithm

In this section, we briefly describe the *RxW* scheduling algorithm, which serves as the basis for our integrated broadcast scheduling and data staging techniques. *RxW* is a practical, low-overhead and scalable algorithm that provides excellent performance across a wide range of settings and performance criteria [AF98]. In this work, we assume that the data items to be broadcast are fixed-length and, thus, we refer to them as *pages*. Scheduling extensions to handle variable length data have been developed elsewhere [VH97], [ST97a], [AM98]; similar extensions for data staging are possible and constitute a part of our future work. As described in [AF98] the best overall scheduling quality can be obtained by an even-handed treatment of hot (popular) and cold (not-so-popular) pages. Based on this intuition, *RxW* schedules a page either because it is very popular or because there is one outstanding request that has waited a long time for that page. At each scheduling decision the RxW algorithm chooses to broadcast the page with the maximal $R \times W$ value where R is the number of outstanding requests for that page and W is the time that the oldest outstanding request for that page has been waiting.

[2] Another approach could be to generate multiple candidates during the scheduling process in order to ensure there is at least one page that can be broadcast immediately. This approach can be useful especially for high-overhead scheduling algorithms. In our case, since the RxW scheduling decision overhead is very low, we take the liberty of rescheduling until a cache hit and therefore use the most current queue state at each scheduling decision.

[3] As soon as the number of outstanding requests drops back below the limit, we switch back to the original scheduling mode.

A. RxW: Implementation

Scheduling is performed at the server in an on-line fashion. The server maintains a service queue that contains a single entry for each page. Each request entry carries all the information that is necessary for making the scheduling decision, namely the number of outstanding request(s) for the page (R) and the arrival time of the oldest of those requests. The arrival time is used to compute the waiting time (W) by simply subtracting from current time at each scheduling decision. The server maintains two sorted lists threaded through the service queue: one based on the number of outstanding requests (referred to as the *R-list*) and the other on the waiting time of the oldest request for that page (referred to as *W-list*). [4]

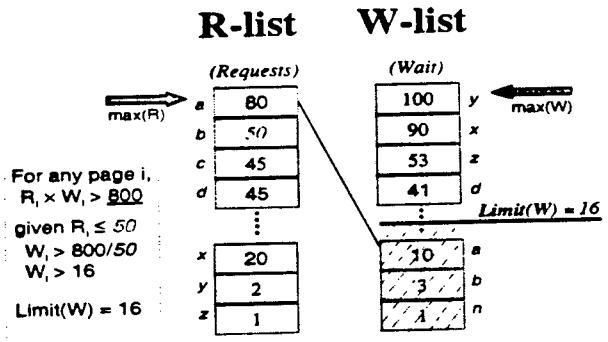


Fig. 3. Pruning the Search Space

These two sorted lists are used to avoid an exhaustive search of the service queue when searching for the page with the maximal $R \times W$ value. The search technique is depicted in Figure 3. The search starts from the entry at the top of the *R-list* (the page with the most outstanding requests) which corresponds to the entry for page *a* in Figure 3. There are 80 outstanding requests for page *a* and the oldest of those pages have been waiting for 10 broadcast ticks (time needed to broadcast a single page). At this point, the *maximum* $R \times W$ value is set to 800. The R value of the following page on the *R-list*, page *b*, is then used to compute a lower limit on the W value for any page that can beat the current maximum $R \times W$ value. Since the entries are sorted in descending order, we know that all entries that are not yet scanned at this point have an R value that is less than or equal to 50. Therefore we know that any entry that could possibly have a higher $R \times W$ value must have a W value greater than $800/50$. This computation enables us to prune the W-list, i.e., we need not go any further beyond this limit on the W-list.

Next, the service queue entry for the page at the top of the *W-list* (the entry with the oldest outstanding request) is examined and similarly, used to place a lower limit on the R value. The algorithm then keeps alternating between the two lists, raising the limit on the other list as appropriate and thereby pruning the search space further. The search stops when the limit is reached on one of the lists. This technique effectively shrinks the search space while still guaranteeing that the search will return the page with the maximum RxW value.

B. RxW.α: A Parameterized Approximation

The overhead of scheduling can be further reduced by relaxing the requirement of broadcasting the page with the highest $R \times W$ value. We have observed that the maximal $R \times W$ valued page typically resides at a much higher location than the one at which the scheduling process ultimately stops; Most of the search is done to guarantee maximality. Using this observation, we developed an approximation-based version of the algorithm. By varying a single parameter α, this approximation ranges in cost from that of the maximal *RxW* algorithm defined above to a constant time algorithm. The approximation algorithm broadcasts the *first* page it encounters whose RxW value is greater than or equal to α times the current *threshold* value. The *threshold* is computed as the running average of the RxW value of the last page broadcast and the previous *threshold*. In cases where no such page is found, the search proceeds within the pruned search space, and the page with the highest $R \times W$ value is broadcast. After each broadcast decision, the *threshold* is updated accordingly.

The setting of the α parameter determines the

[4] Maintaining these two sorted lists is fairly inexpensive; The arrival of a request for a page for which there are currently no outstanding requests results in the creation of a new service queue entry where the *R value* is set to 1 and the arrival time is set to current time. This entry is appended to the tail of the *R-list* and *the W-list*. Thereafter, the arrival of subsequent requests for that page increments the R value of the entry, and relocates the entry in the R-list, but does not impact the entry's position in the W-list. Entries are removed from both lists when the corresponding page is broadcast.

performance tradeoffs among average waiting time, worst case waiting time, and scheduling overhead. The smaller the value of the parameter, the fewer entries are likely to be scanned. [5] For the experiments in this paper we use RxW.90 (i.e., $\alpha = 0.9$) as the broadcast scheduling algorithm. We have observed that in our experimental test-bed configuration, RxW.90 provides a reasonable trade off between scheduling quality (i.e., closeness to the optimal bandwidth allocation) and scheduling overhead (i.e., the time it takes to make a scheduling decision). It should be noted, however, that we have tested our data staging solutions using the full RxW algorithm and its approximations with different α values between 0 and 1. Even though the specific behaviour of the various data staging approaches varies somewhat for different approximation settings, the trends described in this paper hold for all cases tested.

IV. Prototype

As stated in Section I, we have implemented a testbed and used it to study our staging approaches. The prototype is implemented on a cluster of pentium-based workstations running Windows NT 4.0. Each machine has an Intel Pentium Pro 200MHz CPU and 64 MBs of main memory. One of these machines is dedicated as the server. The pages to be broadcast are all initially stored on the server's local disk. The local disk of the server is a fast wide SCSI 4GB Seagate ST32550.

Each machine has two independent Ethernet connections: one for the uplink and one for the downlink. Requests are sent on a 10Mbps uplink and the server broadcasts pages on a 100 Mbps *downlink*. The downlink employs UDP (Unreliable Datagram Protocol) for *multicasting* the data to all of the workstations in the cluster using the IP-multicast support provided with Windows NT 4.0. During experiments, the testbed is isolated from any external network to avoid external network traffic.

The server has two main responsibilities: request processing (queuing new requests), and broadcast management (making scheduling decisions and broadcasting pages). Two threads running on the server perform these two jobs. To ensure that the request arrival rate is fixed across all algorithms, the request processing thread is given top priority. In our prototype server a single CPU handles request processing, and broadcast management. Therefore, the performance numbers that we present include request processing time. In order to minimize this request processing time that appears in the performance results, we avoided using actual messages from the clients over the uplink. Instead, we pre-generate the request pattern for each experiment, and record it on the server disk. The server then uses double buffering to read the requests such that it fills up one buffer as it processes the requests in the other buffer. We monitored the system to verify that the server always finds the second buffer ready when it consumes the requests from the first buffer. All algorithms evaluated experience exactly the same amount of request processing. File buffering of the operating system is disabled throughout the experiments so as not to interfere with the data staging operations.

In the experiments, we first warm up the server cache and then make sure equilibrium is reached before taking measurements. Equilibrium occurs when the number of outstanding requests in the system stabilizes, i.e., the request satisfaction rate converges to the request arrival rate. We use Little's Law [Tri82] to evaluate the average waiting time using the logical service queue length [6].

V. Opportunistic Scheduling

We have introduced the mechanism we use to apply integrated data staging solution in Section II. Recall that we had a limit on the number of outstanding requests and we were switching to opportunistic scheduling mode when this limit is reached, so that only cache-resident pages will be scheduled. In this

[5] Typically the α parameter will be set to a value between 0 and 1. In the limit, setting α to ∞ results in the approximation algorithm behaving identically to the maximal RxW algorithm. At an extreme value of 0, the algorithm simply compares the top entry from both the **R-List** and the **W-list** and chooses the one with the highest RxW value. In this case, the complexity of making a scheduling decision is reduced to $O(1)$, ensuring that broadcast *scheduling* will not become a bottleneck regardless of the broadcast bandwidth, database size, or workload intensity [AF98].

[6] Recall that the length of the physical queue maintained at the server is limited by the number of pages in the database and is much smaller than the logical queue length, i.e. the number of outstanding requests.

section we describe the algorithms we apply for deciding on which cache-resident page to broadcast in order to keep the broadcast channel utilized in the opportunistic scheduling mode. We then evaluate these algorithms in Section V-B.

A. Opportunistic Scheduling Algorithms

We have developed three algorithms for choosing cache-resident pages for broadcast during opportunistic scheduling mode. The first two of these require that scheduler be aware of each page's availability. This requires, the service queue entries maintained by the RxW be extended with a flag that indicates whether or not the corresponding page is cache-resident. This flag is set when the page is brought into cache and cleared when the page is replaced from the cache. Following is the description of the three approaches:

- *Best Cache Resident (OS-BCR):* In this algorithm, we run the scheduler as in the original case. However, the maximum R × W value and the limits on either queue are updated only for the cache resident pages. As a result, we keep track of the best-cache-resident page according to the scheduling algorithm. Note that when we apply opportunistic scheduling, only cache-resident pages are broadcast and therefore only the corresponding entries are deleted from the service queue and then rebuilts up from the bottom of the lists. As a result, cache-resident pages appear at a lower level than they normally would be in the original scheduling mode. Therefore we expect more number of entries to be scanned during the search of opportunistic scheduling, since no matter how high RxW values we observe on non-cache-resident pages, the search stops only when the best cache-resident pages satisfies the stopping condition.
- *Earlier Stop (OS-ES):* This algorithm aims at searching less number of entries while making the opportunistic scheduling decision. It simply runs the scheduler as usual, but keeps track of two broadcast candidates: best cache-resident page and overall-best page (including pages that are not cache-resident). The search stops when the overall-best page meets the stopping condition, rather than the best cache-resident page. We refer to this case where the scheduler stops as if all pages are in the cache as the original application of the algorithm. At the end of this original application, the best cache-resident page (among the ones searched upto this point), if any, is selected for broadcast. If no cache-resident page is encountered at this point, the search continues until the *first* cache-resident page. Therefore less number of entries need to be scanned during the search using OS-ES. Note that the algorithm does not guarantee that the *best* cache-resident page (among all that are available) according to the scheduling criteria is broadcast.
- *Scan Cache (OS-SC):* The intuition behind the algorithm is to minimize the overhead involved in selecting a cache-resident page to broadcast. This algorithm does not use the RxW scheduling algorithm except for the initial check to see if the original application yields a cache hit. Otherwise it basically cycles through the pages in the cache using a pointer, referred to as *nextToBroadcast*, initially set to the page at the top of the cache. The algorithm simply broadcasts the page pointed by *nextToBroadcast* if there are any outstanding requests for this page. Otherwise the pointer is advanced to the next page in cache until one with outstanding requests is found and broadcast. After the broadcast, the pointer is advanced to the next page in order to determine the broadcast candidate in the next opportunistic scheduling. To avoid unexpected behavior, we also advance the pointer when the pointed page is replaced in the cache.

B. Opportunistic Scheduling: Evaluation

In this section we experiment our approaches on the test-bed we have implemented as described in Section IV. We first describe the workload we applied in this and subsequent experiments. We then report the results of the experiment.

The workload we used for the majority of the experiments is based on a Zipf distribution with θ set to 1 [Knu81]. The Zipf distribution is such that

$$p_i = \frac{1}{i^\theta \sum_{j=1}^{N} \frac{1}{j^\theta}}$$

where p_i is the probability of accessing page i, N is the size of the database and θ is the skewness parameter. The database consists of 10000 pages. All pages are 16K and disk-resident. At most 100 concurrent asynchronous I/O requests are allowed during the experiments. The cache size is varied between 5% and 100% of the database. LRU cache replacement policy is used throughout these experiments. The approaches we have developed are aimed at large-scale systems with many thousands of clients. We, therefore, stress-test our prototype at an arrival rate of 1000 requests/sec. We have also tried lower and higher rates and we observed similar results for opportunistic scheduling algorithms. We describe the effects on other data staging approaches in corresponding sections. We now proceed with the evaluation of opportunistic scheduling.

As already discussed in Section V, blocking on a cache miss is not favorable at all. To see how much of a performance penalty such ignorance of the data staging problem would bring, we compare the performance of opportunistic scheduling algorithms against such a case where data staging is completely ignored. Figure 4 shows the performance results of this experiment. In this experiment, the average waiting time is measured as the cache size increases from 5% of the database to 100% of the database. In Figure 4 SYNCH refers to the case where the server issues a synchronous I/O request for cache misses and stalls until the required page is broadcast. As expected opportunistic scheduling (the bottom most tree curves curves) provides orders of magnitude improvement across all cache sizes. For instance, at 4000 page cache size, all opportunistic scheduling approaches have 17 times better performance than SYNCH. The improvement obtained by opportunistic scheduling is due to the better bandwidth utilization. For instance, at 4000 page cache size opportunistic scheduling uses 98% of the bandwidth that could be used (if all pages were in the cache). In contrast SYNCH uses only 22% of the bandwidth that could be used. Note that the relative bandwidth usage would be even more significant for latencies higher than secondary storage, i.e., when data needs to be retrieved from remote sites. All curves converge at 10000 page cache size, since all pages are in the cache and there is not any cache miss that will differentiate the behavior. At this point the performance is fully determined by the scheduling algorithm used which is the same for all three.

Next we take a closer look at the opportunistic scheduling algorithms. Figure 5 zooms up the bottom left corner of Figure 4. Here, we see at OS-SC gives the best performance across all ranges. OS-ES gives the worst performance across all ranges except for 500 page cache. All algorithms converge at 4000 page cache, because then the asynchronous miss queue is not full and therefore opportunistic scheduling is not applied for any algorithm. In the figure, we observe a significant performance difference between the algorithms at 3000 page cache, i.e., OS-SC is 2.3 times better than OS-ES at this point. The performance differences come from two factors: the broadcast efficiency and the broadcast error made. The trade off between these factors define the performance of an algorithm.

For instance, OS-ES's poor performance is mostly due to the scheduling error made, even though OS-ES has a quite good broadcast efficiency. Figure 6 plots the bandwidth usage for the three algorithms measured for the same experiment. The bandwidth usage shown does not include any accompanying data flow information, such as UDP packet headers etc. Bandwidth is measured as the total number of data bytes broadcast per second. Also note that the maximum bandwidth usage is limited by 59Mbps, even if all pages are in the cache due to NT system overhead. In other words, OS-ES can use at least 85% of the available bandwidth in the whole range of measured cache sizes. On the other hand, OS-ES results in a very poor broadcast schedule. It actually converges to always selecting the cache-resident page with the most outstanding requests during opportunistic scheduling mode. This kind of over favoring hot pages has already been shown to provide a poor performance [AF98]. Figure 7 plots the bandwidth allocation error measured during the experiment. The bandwidth allocation error metric is computed by comparing the resulting bandwidth allocation generated by the algorithm versus the optimal bandwidth allocation that is defined in [DAW86]. The optimal allocation must be done in relative square root ratios of page access probabilities. We measure the rate at which each page is

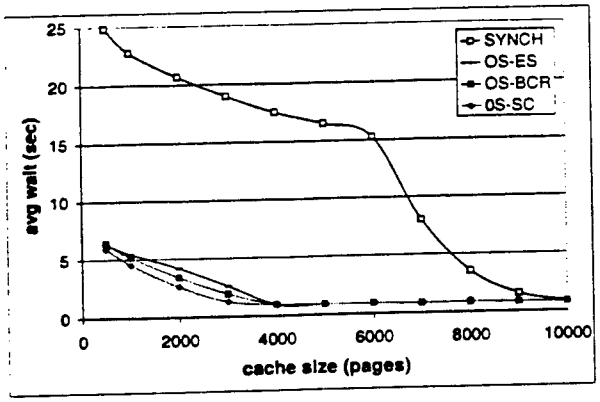


Fig. 4. Average Waiting Time

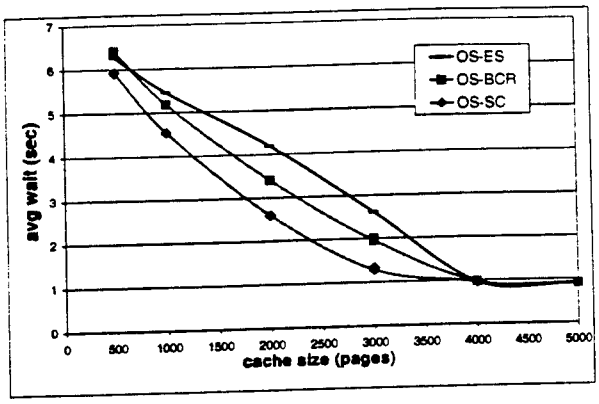


Fig. 5. Average Wait - Opportunistic Scheduling Only

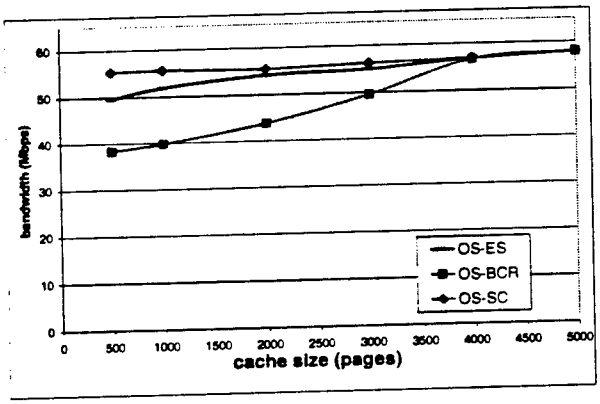


Fig. 6. Scheduling Efficiency

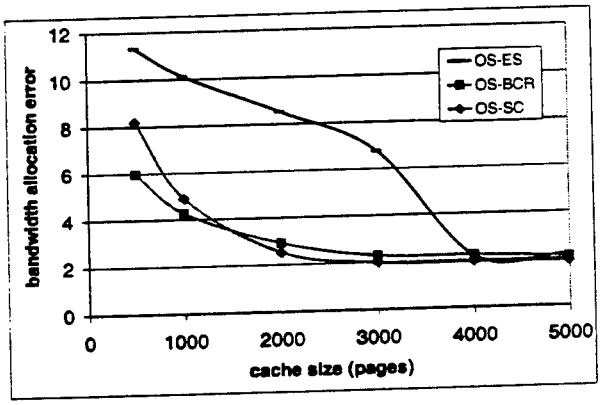


Fig. 7. Scheduling Error

broadcast and average the percentage of the absolute error made when compared to this optimal case. As shown in Figure 7 OS-ES results in the highest bandwidth allocation error among all three algorithms.

Going back to Figure 5, OS-BCR performs better than OS-ES since, OS-BCR makes better broadcast decisions. As shown in Figure 7 OS-BCR makes significantly less error when compared to OS-ES. Therefore OS-BCR performs better than OS-ES, even though OS-BCR has worse broadcast efficiency, i.e., slower, than OS-ES since it scans more service queue entries to decide on the cache-resident page to broadcast. This is due to the fact that the scheduling overhead is increased significantly when the scheduler is restricted to choose a cache resident page. For instance, the number of entries that need to be scanned increases 2000 fold for a cache size of 500 pages when compared to OS-ES. In Figure 6 OS-BCR is shown to be the most inefficient algorithm among all three.

Finally, OS-SC gives the best performance in Figure 5, because it wins on both factors. OS-SC has the minimum overhead among all three algorithms, and therefore yields the best bandwidth usage as shown in Figure 6. In addition, OS-SC yields a pretty good broadcast schedule. The cache-resident pages that are broadcast during opportunistic scheduling are only a small subset of the pages that are of interest to the client population, and therefore a uniform broadcast of pages with outstanding requests does a pretty good job of scheduling in the overall.

These results demonstrate that broadcast efficiency is just one of many aspects that contribute to performance in a real system, and that the sum of the effects of these different components are what determine overall performance. As described in [AF98], the trade off between different components was in fact the motivation behind the approximate versions of

RxW. Opportunistic scheduling solves the broadcast efficiency problem when compared to synchronous requests, however it converts this efficiency problem to broadcast error problem.

VI. Server Cache Management for Broadcast-Based Servers

For a highly skewed distribution such as the Zipf, *RxW* will broadcast individual hot pages more than their colder counterparts, but in the overall, significant bandwidth will be given to cold pages. In our experiments we have observed that approximately 1/3 of the bandwidth is expended broadcasting the top 10% hottest pages, with the remaining 2/3 going to the colder pages. This implies that for small cache sizes, LRU is not very useful. LRU has the well-known property that once a page is access - *broadcast* in our case - it will be placed at the top of the LRU stack and then it has to travel all the way down before it will be replaced from the cache. This implies that for small cache sizes: 1) Cold pages will always result in a cache miss when LRU is employed; since the low frequency of cold page broadcasts will make them the *least recently used* page (they travel all the way to the tail of LRU stack) before they are scheduled again. 2) It is not only the cold pages, but also the hot pages that will suffer in this case; since the high number of cold pages being broadcast can force hot pages to be flushed off the cache before they are scheduled for broadcast again.

Fortunately it is possible to replace the server cache replacement policy with one that better matches the broadcast scheduling algorithm used. Alternatives to LRU that avoid the problem of cold pages replacing hot pages have been proposed (e.g., LRU-K [OOW93] and 2Q [JS94]). These policies maintain past reference history for items that are no longer in the cache and use it to distinguish cold pages from hot ones. In our environment, however, we have a unique advantage, namely, that the *RxW* algorithm already provides valuable information that can reliably be used to distinguish hot pages from cold, without the need to store additional access history. In the following subsection we investigate the technique that uses this information.

A. LRU With Love/Hate Hints (LH)

In this section we describe the algorithm that we use to improve server's cache management. Recall that *RxW* aims to provide a balanced treatment of hot and cold pages. More explicitly, *RxW* broadcasts a page either if it is popular enough or if it has been waited for long enough. Hot pages are more likely to be broadcast because they have a large number of outstanding requests and hence, will be high on the *R-list* when they are scheduled for broadcast. On the other hand, cold pages are likely to have accumulated some waiting time before they are scheduled and therefore will be high on the W-list. Thus, due to the data structure used by *RxW* it is possible to distinguish between the popularity of pages and treat these pages accordingly. Pages that appear to be hot at the time they are chosen for broadcast are tagged with a "love" hint and placed at the top of the LRU stack, while those that appear to be cold are tagged with a "hate" hint and put at the bottom of the LRU stack where they are likely to be chosen as replacement victims. We refer to this extension of LRU as "LRU-LH" or simply "LH". With "LH" we expect the cache to converge to a state where it only keeps popular pages.

To decide if a page chosen for broadcast should be considered hot (i.e., marked with love-hint) the page must meet the following tests:

1. In the scheduler's alternating search of the *R* and *W* lists the page must be encountered on the *R-list before* it is encountered on the *W-list*.
2. The page must appear in the top *hot range* pages of the *R-list*, and there must not be any pages with the same number of requests that lie beyond this range.

The first test ensures that the page is higher on the *R-list* than on the *W-list*. This requirement is satisfied by simply updating the current maximum *RxW* with a only a larger value, excluding equal values. As a result once a page is selected on the *R-list* it is guaranteed that it has not been scanned on the *W-list* yet and the page is a candidate for a hot page. The second test aims to reserve the cache for the hottest pages that can fit. The special handling for "ties" is intended to avoid over-committing the cache in the case that many pages have the same *R* value. The size of the *hot range* is set to be a fraction of the

size of the cache and the number of outstanding page requests as

$$cacheSize \times \frac{entryCnt}{dbSize}$$

where *entryCnt* is the total number of pages queued for broadcast at the time of scheduling decision, and *dbSize* is the total number of unique pages requested by the client population [7].

B. Caching: Evaluation

In this section we investigate possible improvements by more suitable server cache management policies. The workload parameters are as described in Section V-B. In this experiment, we take the best opportunistic scheduling algorithm of Section V-B, namely OS-SC and replace the cache replacement policy of LRU with LH. To evaluate the benefits of LH we compare it with LRU and an idealized (i.e., impractical) algorithm called PCACHE. This latter algorithm uses perfect knowledge of the data access distribution and keeps the pages with the highest access probabilities in cache at all time. PCACHE demonstrates the ideal case LH is aiming at converging to.

We use the same workload as described in Section V-B for this experiment. The average waiting time for the three caching policies is shown in Figure 8. Note that, the curve labeled as LRU is the same curve labeled as OS-SC in Figure 5. In all three approaches OS-SC alternative is being used for opportunistic scheduling. On the x-axis, the cache size is varied up to 5000 pages, and the y-axis shows the average waiting time for the algorithms. As can be seen in the figure, LRU provides the worst performance across the entire range of cache sizes. LH performs significantly better than LRU. For instance, at 2000 page cache, LH is 2.8 times better than LRU. LH is almost as good as PCACHE across the entire range. This suggests that LH is very successful in converging to the behavior of keeping the cache with the top most popular pages. This observation is further supported by the hit rates shown in Figure 9 for the same experiment. We observe that LH improves the hit rate almost as much as PCACHE does. This proves the ability to distinguish between hot pages and cold pages through the use of the RxW algorithm. Due to the increased hit rate, opportunistic scheduling is used less frequently and the optimal decision made by RxW.90 is realized more often. Going back to Figure 8 we see all algorithms converge for hit rates over 0.5. [8] This performance is obtained when the majority of the missed pages have only one or two requests pending and those requests have already accumulated a high waiting time (around 4 seconds). Therefore an additional disk I/O latency does not impact the overall performance a lot. This small penalty of cache misses is easily paid off by the support of opportunistic scheduling.

The main result of Figure 8 is that with LH a cache size of 20% of the database is enough for perfect performance. This suggests that LH needs half the cache size LRU would need for this skewed workload. This improvement is especially important when we consider large database sizes being accessed with a skewed distribution.

VII. Reducing I/O Latency

The final front of our attack to data staging problem is to bring the pages to the cache before they are actually scheduled for broadcast. The trick here is to be able to predict broadcast decisions. As with caching, we exploit properties of the *RxW* algorithm to make such predictions.

A. Prefetching

We focus our prefetching efforts on predicting which *cold* pages are likely to be chosen for broadcast. There are two reasons behind this choice. First, prefetching requires prediction based on the current state of the service queue. The top of the *W-list* is stable, it can change only when a page is broadcast. In contrast, the top of the *R-list* which is much more volatile. It can change for every request that arrives at the server. Thus, the prediction of pages that are likely to be chosen due to their *W* value (i.e., cold pages) is more reliable than for hot pages. Second, the hit rate on cold pages is expected to be very low,

[7] In general, if the database size is not known a-priori (e.g., when pages reside on remote sites), an estimate by observing the pages requested from the server can be used.

[8] Note that the hit rates in Figure 9 are not yet at 1 since the cache can keep only half of the pages in the database.

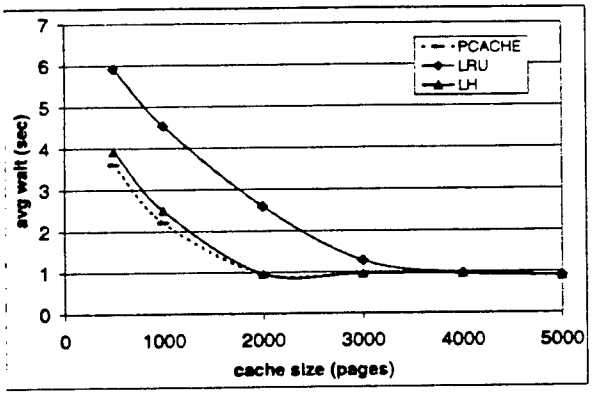


Fig. 8. Average Wait Time

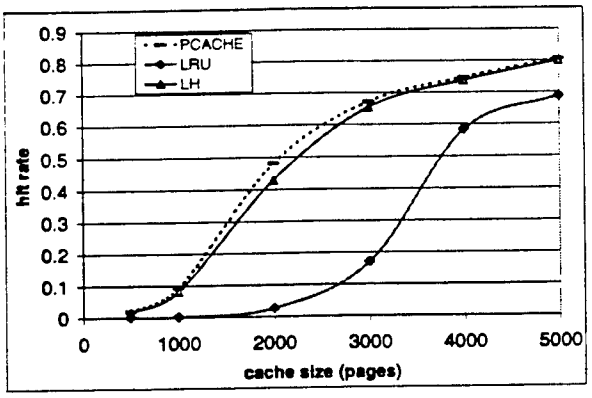


Fig. 9. Hit Rate

and the LH caching policy aims at effectively keeping hot pages in cache. We apply a division of labor between the two approaches used in an integrated design. Caching has the responsibility of keeping track of the pages at the top of the R-list, and prefetching has the responsibility of keeping track of the pages at the top of the the W-list.

The main idea of prefetching is to make sure that all pages within a defined range at the top of the *W-list* are either already in the cache or are in the process of being prefetched. The size of the range is a parameter called *prfWindow*. A buffer of *prfWindow* pages is reserved in the cache (i.e., taken out of the LH-managed space) and background threads are used to prefetch pages into this buffer. When a page from the prefetch buffer is broadcast (and hence, its entry removed from the *W-list*), the prefetch of a new page is initiated. If a page is scheduled for broadcast while it is in the process of being prefetched, the page is broadcast as soon as it arrives.

B. Prefetching: Evaluation

In this experiment we evaluate the effectiveness of prefetching using the workload as described in Section V-B. Figure 10 shows the average wait time with prefetching (labeled "PRF" on the graph) and without (labeled "LH"). For both cases, Scan Cache Opportunistic Scheduling (OS-SC) and LRU with Love/Hate hints (LH) is being applied. The difference comes from the use of prefetching. The total cache size including the buffer for the pages to prefetch is varied from 500 to 5,000 pages. These results were generated with a *prfWindow* of 250. This value is chosen according to our experiments varying the prefetch window at the cache size of 2000 pages.

In the first experiment, we apply data staging to disk-resident data. The limited effectiveness of prefetching in this case can be seen in Figure 10. In Figure 10 we see that when prefetching is added to LH there is little or no improvement when compared to LH alone. This behavior arises, since the average wait for pages is much higher than the time it takes to read a page from disk (even if the disk is highly utilized). The cost of the I/O does not significantly contribute to the time it takes for a user request to be satisfied, thus prefetching is not very helpful in reducing the latency of secondary storage. The two curves curverge when the hit rate of the server is already above the tolerable limit of 0.5 as described in Section VI-B, i.e., when additional disk I/O does not impact the waiting time significantly.

In Figure 11, we plot the results for higher latencies for page retrieval. In this case we use a cache size of 2000 pages and we gradually increase the latency of retrieving pages. As the latency increases we see a great advantage coming from prefetching. At around 1 second, PRF results 95% improvement in terms of the waiting time compared to that would be achieved when prefetching is not used. The performance boost comes from the improved scheduling quality when prefetching is used. We observe the bandwidth allocation error curves for the same experiment increase in a similar behaviour as shown for average waiting time. The two curves converge at latencies around 25 seconds, where latency becomes the dominating

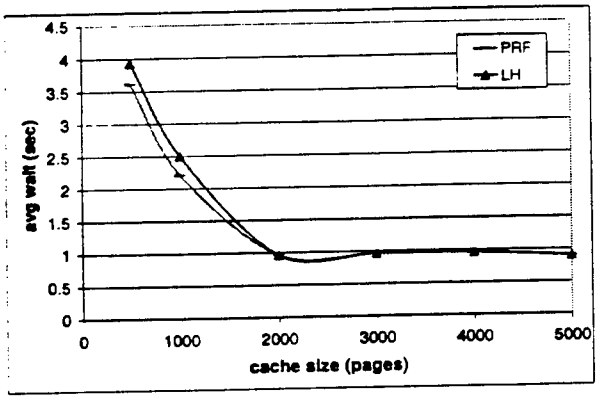


Fig. 10. Average Wait (Disk Resident Data)

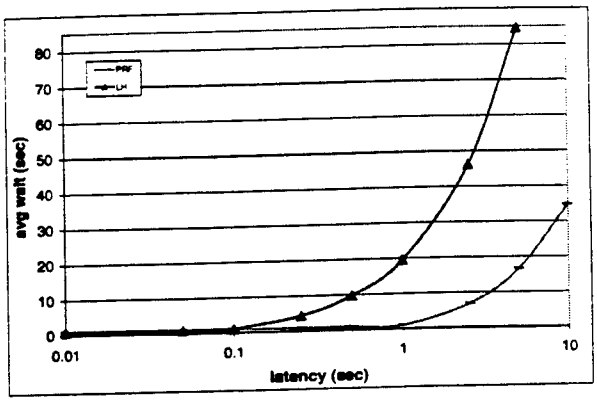


Fig. 11. Average Wait varying Latency

factor.

VIII. Related Work

As stated previously, there has been much work on developing scheduling algorithms for on demand data broadcasting [DAW86], [DW88], [Won88], [VH96], [ST97b], [AF98] and that all of this work with the exception of Dykeman et al. [DW88] ignored the issue of data staging. The Dykeman et al. work was performed assuming an environment that differed substantially from the one studied here. First, the context of that work was *Teletext* systems, which had much lower bandwidths and hence, much smaller databases to broadcast. Thus, the solutions in that earlier paper used a very expensive scheduling and cache replacement algorithm that would not scale for the large-scale systems with very large databases. Second, some of the approaches in [DW88] were based on assumptions about detailed control over disk devices, that are not applicable to today's commodity disk drives and controllers, and do not address the problem of data residing on remote sites. Finally, the Dykeman et al. study was done using simulation so much of the overhead and contention that arises in a real system was not considered. Despite these differences, this study provides a powerful insight to pioneer data staging problem and our results have some common conclusions: speeding up the rate at which requested pages are retrieved is of top priority for performance even if this is at the expense of not being able to retrieve the page with the highest priority immediately.

The result that the optimal broadcast bandwidth allocation is in proportion to the ratios of the square roots of the page access probabilities was shown in [DAW86] and [AW87]. Recall that this property of broadcast plays a key role in our result that low overhead Opportunistic Scheduling approaches can have a poor performance due to poor bandwidth utilization.

In the more general context, data staging has also been studied for multimedia systems. Ozden et al. [ORS96] studied buffer replacement algorithms for multimedia storage systems that exploit the large file sizes and sequential access found in many multimedia applications. Aggarwal et al. [AWY96] have studied scheduling algorithms for Video-On-Demand systems, and proposed a heuristic that uses the number of outstanding requests per page and the broadcast history followed up per page. This study also ignores the data staging problem. In [FD95], prefetching was shown to be an effective performance enhancer for video-on-demand systems. The success of prefetching in this study, however, is mainly based on the sequential access of video files. Prefetching has been used together with caching to reduce access latencies in many other contexts. For example, Patterson et al.'s informed prefetching study [PGG+95] has shown that prefetching using hints from applications is an effective way of exploiting IO concurrency. As stated previously, while prefetching is also helpful in the on-demand broadcast setting especially for high latencies, the need for it is obviated for secondary storage latencies by the ability to do background "post-fetching" with little penalty in additional path length for page access

and little degradation of the quality of the broadcast schedule.

IX. Conclusion

We have presented a data delivery scheme that is especially good for large client populations with a high overlap in interest. In this scheme, the server receives individual pull requests from clients and broadcasts the results. We have shown that a real implementation with a large database and a fast broadcast channel must take data staging concerns seriously in order to achieve reasonable performance.

Our basic approach integrates broadcast scheduling and data staging in a novel way. We use love/hate hints derived from the scheduling data structures to guide the caching of popular pages. Beyond that, we have shown that, while it is tempting to use prefetching to improve the performance of the cache, it is more effective to use a "post-fetching" technique that we call opportunistic scheduling, for secondary storage latencies. The best results , for secondary storage latencies occur when we simply keep the broadcast filled with decent items instead of worrying about sending the best items which might be hard to obtain because of staging problems. For higher latencies, however, prefetching proved to be very effective.

We have implemented a prototype system on Windows NT 4.0 to show that the data staging algorithms that we propose dramatically reduce the penalty of data retrieval latency.

In the future, we plan to focus more on wide-area systems in which the data items of interest may be located on other machines which, like a disk, introduce additional latency. We will also look at problems introduced by variable length data items and by stream-oriented data with ordering constraints (e.g., video).

References

[AF98] D. Aksoy and M. Franklin. Scheduling for large-scale on-demand data broadcasting. In *Proceedings of IEEE INFOCOM*, March 1998.

[AF99] D. Aksoy and M. Franklin. RxW: A scheduling approach to large scale on-demand broadcast. In *Conditionally accepted for IEEE/ACM Transactions on Networking*, 1999.

[AM98] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proc. of Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom)*, Dallas, TX, 1998.

[AW87] M. H. Ammar and J. W. Wong. On the optimality o cyclic transmissions in teletext systems. *IEEE Transactions on Communications*,35(1):68–73, December 1987.

[AWY96] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *The Third IEEE International Conference on Multimedia Computing and Systems*, Hiroshima, Japan, June 1996.

[Cha98] ISP Channel. Your high speed connection to the internet. http://www.ispchannel.com, 1998.

[Cyb99] CyberStream. High speed internet access; CyberStream on VSAT. http://speedus.com, 1999.

[Dat99] Cable Data. An information service of kinetic strategies inc. cabledatacomnews.com, 1999.

[DAW86] H.D. Dykeman, M. Ammar, and J.W. Wong. Scheduling algorithms for videotex systems under broadcast delivery. In *IEEE International Conference on Communications*, pages 1847–1851, Toronto, Canada, 1986.

[Dir96] Hughes network systems, direcpc home page. http://www.direcpc.com, 1996.

[DW88] H.D. Dykeman and J.W. Wong. A performance study of broadcast information delivery systems. In *Proc. IEEE Infocom*, New Orleans, LA, 1988.

[FD95] C. S. Freedman and D. J. DeWitt. The spiffi scalable video-on-demand system. In *SIGMOD*, San Jose, CA, 1995.

[Hom98] @Home network home page. URL: http://www.home.net, February 1998.

[Hyb] Hybrid. High-speed data access for teleworkers. http://www.hybrid.com.

[Int98] IntelliQuest. Information solutions for global technology marketing. http://www.intelliquest.com, 1998.

[JS94] T. Johnson and D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 439–450, Santiago de Chile, Chile, September 1994.

[Knu81] D. Knuth. *The art of Computer Programming - Volume III.* Addison-Wesley, 1981.

[mee92] IETF meeting. Multicast backbone. http://info.brl.mil/ARL-Directorates/ASHPC/HPCD/MBONE/, 1992.

[OOW93] E.J. O'Neil, P.E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 297–306, Washington, DC, may 1993.

[ORS96] B. Ozden, R. Rastogi, and A. Silberschatz. Buffer replacement algorithms for multimedia storage sys-

tems. In *IEEE International Conference on Multimedia Computing and Systems*, June 1996.

[PGG+95] R.H. Patterson, G.A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proc. of the 15th Symp. On Operating System Principles*, Dec. 1995.

[ST97a] C.J. Su and L. Tassiulas. Broadcast scheduling for distribution of information items with unequal length. In *In Proc. 31th Conf. on Information Sciences and Systems (CISS'97)*, Kobe, Japan, March 1997.

[ST97b] C.J. Su and L. Tassiulas. Broadcast scheduling for information distribution. In *Proc. IEEE INFOCOM*, 1997.

[Tel94] Teldesic corporation with the federal communications commission. Application of Teledesic Corporation for a Low Earth Satellite System in the Domestic and International Fixed Satellite Services, March 1994.

[Tri82] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Application*. Prentice-Hall Inc, 1982.

[VH96] N.H. Vaidya and S. Hameed. Data broadcast in assymetric wireless environments. In *Proc. of Workshop on Satellite-based Information Services (WOSBIS)*, New York, November 1996.

[VH97] N.H. Vaidya and S. Hameed. Log-time algorithms for scheduling single and multiple channel data broadcast. In *Proc. of Workshop on Satellite-based Information Services (WOSBIS)*, Budapest, Hungary, September 1997.

[Web99] Cable Web. An information service of kinetic strategies inc. cableweb.com, 1999.

[Won88] J.W. Wong. Broadcast delivery. *Proc. of IEEE*, 76(12):1566–1577, December 1988.

Appendix C

Research in Data Broadcast and Dissemination

Research in Data Broadcast and Dissemination

Demet Aksoy[2], Mehmet Altinel[2], Rahul Bose[1], Ugur Cetintemel[2],
Michael Franklin[2], Jane Wang[1] and Stan Zdonik[1]

[1] Department of Computer Science, Brown University, Providence, RI 02912
[2] Department of Computer Science, University of Maryland, College Park, MD 20742

1 Introduction

The proliferation of the Internet and intranets, the development of wireless and satellite networks, and the availability of asymmetric, high-bandwidth links to the home, have fueled the development of a wide range of new "dissemination-based" applications. These applications involve the timely distribution of data to a large set of consumers, and include stock and sports tickers, traffic information systems, electronic personalized newspapers, and entertainment delivery. Dissemination-oriented applications have special characteristics that render traditional client-server data management approaches ineffective. These include:

- tremendous scale.
- a high-degree of overlap in user data needs.
- asymmetric data flow from sources to consumers.

For example, consider a dissemination-oriented application such as an election result server. Typically, such applications are implemented by simply posting information and updates on a World Wide Web server. Such servers, however, can and often do become overloaded, resulting in the inability for users to access the information in a timely fashion. We argue that such scalability problems are the result of a mismatch between the data access characteristics of the application and the technology (in this case, HTTP) used to implement the application. HTTP is based on a request-response or RPC, unicast (i.e., point-to-point) method of data delivery, which is simply the wrong approach for this type of application.

Using request-response, each user sends requests for data to the server. The large audience for a popular event can generate huge spikes in the load at servers, resulting in long delays and server crashes. Compounding the situation is that users must continually *poll* the server to obtain the most current data, resulting in multiple requests for the same data items from each user. In an application such as an election server, where the interests of a large part of the population are known *a priori*, most of these requests are unnecessary.

The use of unicast data delivery likewise causes problems in the opposite direction (from servers to clients). With unicast the server is required to respond individually to each request, often transmitting identical data. For an application with many users, the costs of this repetition in terms of network bandwidth and server cycles can be devastating.

To address the particular needs of dissemination-based applications, we are developing a general framework for describing and constructing Dissemination-Based Information Systems (DBIS). The framework incorporates a number of data delivery mechanisms and an architecture for deploying them in a networked environment. The goal is to support a wide range of applications across many varied environments, such as mobile networks, satellite-based systems, and wide-area networks. By combining the various data delivery techniques in a way that matches the characteristics of the application and achieves the most efficient use of the available server and communication resources, the scalability and performance of dissemination-oriented applications can be greatly enhanced.

In this paper, we provide an overview of the current status of our DBIS research efforts. We first explain the framework and then describe our initial prototype of a DBIS toolkit. We then focus on several research results that have arisen from this effort.

2 The DBIS Framework

There are two major aspects of the DBIS framework.[3] First, the framework incorporates a number of different options for data delivery. A taxonomy of these options is presented in Section 2.1 and the methods are further discussed in Section 2.2. Secondly, the framework exploits the notion of *network transparency*, which allows data delivery mechanisms to be mixed-and-matched within a single application. This latter aspect of the framework is described in Section 2.3.

2.1 Options for Data Delivery

We identify three main characteristics that can be used to describe data delivery mechanisms: (1) push vs. pull; (2) periodic vs. aperiodic; and (3) unicast vs. 1-to-N. Figure 1 shows these characteristics and how several common mechanisms relate to them.

Client Pull vs. Server Push - The first distinction we make among data delivery styles is that of "pull vs. push". Current database servers and object repositories support clients that explicitly send requests for data items when they require them. When a request is received at a server, the server locates the information of interest and returns it to the client. This *request-response* style of operation is *pull-based* — the transfer of information from servers to clients is initiated by a client pull. In contrast, push-based data delivery involves sending information to a client population in advance of any specific request. With push-based delivery, the server initiates the transfer.

The tradeoffs between push and pull revolve around the costs of initiating the transfer of data. A pull-based approach requires the use of a backchannel for

[3] Parts of this section have been adapted from an earlier paper, which appeared in the 1997 ACM OOPSLA Conference [Fran97].

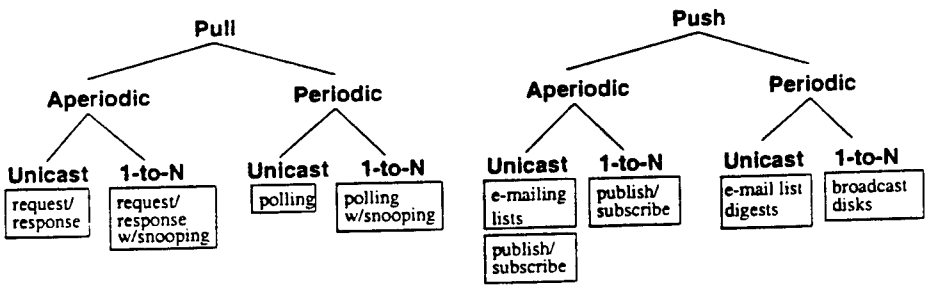


Fig. 1. Data Delivery Characteristics

each request. Furthermore, as described in the Introduction, the server must be interrupted continuously to deal with such requests and has limited flexibility in scheduling the order of data delivery. Also, the information that clients can obtain from a server is limited to that which the clients know to ask for. Thus, new data items or updates to existing data items may go unnoticed at clients unless those clients periodically poll the server.

Push-based approaches, in contrast, avoid the issues identified for client-pull, but have the problem of deciding which data to send to clients in the absence of specific requests. Clearly, sending irrelevant data to clients is a waste of resources. A more serious problem, however, is that in the absence of requests it is possible that the servers will not deliver the specific data that are needed by clients in a timely fashion (if ever). Thus, the usefulness of server push is dependent on the ability of a server to accurately predict the needs of clients. One solution to this problem is to allow the clients to provide a *profile* of their interests to the servers. *Publish/subscribe* protocols are one popular mechanism for providing such profiles.

Aperiodic vs. Periodic - Both push and pull can be performed in either an aperiodic or periodic fashion. Aperiodic delivery is *event-driven* — a data request (for pull) or transmission (for push) is triggered by an event such as a user action (for pull) or data update (for push). In contrast, periodic delivery is performed according to some pre-arranged schedule. This schedule may be fixed, or may be generated with some degree of randomness.[4] An application that sends out stock prices on a regular basis is an example of periodic push, whereas one that sends out stock prices only when they change is an example of aperiodic push.

[4] For the purposes of this discussion, we do not distinguish between fixed and randomized schedules. Such a distinction is important in certain applications. For example, algorithms for conserving energy in mobile environments proposed by Imielinski et al. [Imie94b] depend on a strict schedule to allow mobile clients to "doze" during periods when no data of interest to them will be broadcast.

Unicast vs. 1-to-N - The third characteristic of data delivery mechanisms we identify is whether they are based on unicast or 1-to-N communication. With unicast communication, data items are sent from a data source (e.g., a single server) to one other machine, while 1-to-N communication allows multiple machines to receive the data sent by a data source. Two types of 1-to-N data delivery can be distinguished: multicast and broadcast. With multicast, data is sent to a specific subset of clients. In some systems multicast is implemented by sending a message to a router that maintains the list of recipients. The router reroutes the message to each member of the list. Since the list of recipients is known, it is possible to make multicast reliable; that is, network protocols can be developed that guarantee the eventual delivery of the message to all clients that should receive it. In contrast, broadcasting sends information over a medium on which an unidentified and unbounded set of clients can listen. This differs from multicast in that the clients who may receive the data are not known *a priori*.

The tradeoffs between these approaches depend upon the commonality of interest of the clients. Using broadcast or multicast, scalability can be improved by allowing multiple clients to receive data sent using a single server message. Such benefits can be obtained, however, only if multiple clients are interested in the same items. If not, then scalability may actually be harmed, as clients may be continually interrupted to filter data that is not of interest to them.

2.2 Classification of Delivery Mechanisms

It is possible to classify many existing data delivery mechanisms using the characteristics described above. Such a classification is shown in Figure 1. We discuss several of the mechanisms below.

Aperiodic Pull - Traditional request/response mechanisms use aperiodic pull over a unicast connection. If instead, a 1-to-N connection is used, then clients can "snoop" on the requests made by other clients, and obtain data that they haven't explicitly asked for (e.g, see [Acha97, Akso98]).

Periodic Pull - In some applications, such as remote sensing, a system may periodically send requests to other sites to obtain status information or to detect changed values. If the information is returned over a 1-to-N link, then as with request/response, other clients can snoop to obtain data items as they go by. Most existing Web or Internet-based "push" systems are actually implemented using Periodic Pull between the client machines and the data source(s) [Fran98].

Aperiodic Push - Publish/subscribe protocols are becoming a popular way to disseminate information in a network [Oki93, Yan95, Glan96]. In a publish/subscribe system, users provide information (sometimes in the form of a profile) indicating the types of information they wish to receive. Publish/subscribe is push-based; data flow is initiated by the data sources, and is aperiodic, as there is no predefined schedule for sending data. Publish/subscribe protocols are inherently 1-to-N in nature, but due to limitations in current Internet technology, they are often implemented using individual unicast messages to multiple clients. Examples of such systems include Internet e-mail lists and some existing "push" systems on the Internet. True 1-to-N delivery is possible through technologies

such as IP-Multicast, but such solutions are not universally available across the Internet.

Periodic Push - Periodic push has been used for data dissemination in many systems. An example of Periodic Push using unicast is Internet mailing lists that send out "digests" on a regular schedule. For example, the Majordomo system allows a list manager to set up a schedule (e.g., weekly) for sending digests. Such digests allow users to follow a mailing list without being continually interrupted by individual messages. There have also been many systems that use Periodic Push over a broadcast or multicast link. These include TeleText [Amma85, Wong88], DataCycle [Herm87], Broadcast Disks [Acha95a, Acha95b] and mobile databases [Imie94b].

2.3 Network Transparency

The previous discussion has focused primarily on different modes of data delivery. The second aspect of the DBIS framework addresses how those delivery modes are used to facilitate the efficient transfer of data through the nodes of a DBIS network. The DBIS framework defines three types of nodes:

1. *Data Sources*, which provide the base data to be disseminated.
2. *Clients*, which are net consumers of information.
3. *Information Brokers*, (or agents, mediators, etc.), which acquire information from other sources, possibly add value to that information (e.g., some additional computation or organizational structure), and then distribute this information to other consumers.

Brokers are the glue that bind the DBIS together. Brokers are middlemen; a broker acts as a client to some number of data sources, collects and possibly repackages the data it obtains, and then functions as a data source to other nodes of the system. By creating hierarchies of brokers, information delivery can be tailored to the needs of many different users.

The ability of brokers to function as both clients and data sources provides the basis for the notion of Network Transparency. Receivers of information cannot detect the details of interconnections any further upstream than their immediate predecessor. Because of this transparency, the data delivery mechanism used between two or more nodes can be changed without requiring changes to the data delivery mechanisms used for other communication in the DBIS. For example, suppose that node B is pulling data values from node A on demand. Further, suppose that node C is listening to a periodic broadcast from node B which includes values that B has pulled from A. Node C will not have to change it's data gathering strategy if A begins to push values to B. Changes in links are of interest only to the nodes that are directly involved. Likewise, this transparency allows the "appearance" of the data delivery at any node to differ from the way the data is actually delivered earlier in the network. This in turn, allows the data delivery mechanisms to be tailored for a given set of nodes. For example, a

broker that typically is very heavily loaded with requests could be an excellent candidate for a push-based delivery mechanism to its clients.

Current Internet "push" technology, such as that provided by PointCast [Rama98] provide an excellent example of network transparency in action. To the user sitting at the screen, the system gives the impression of using aperiodic push over a broadcast channel. Due to current limitations of the Internet, however, that data is actually brought over to the client machine using a stream of periodic pull requests, delivered in a unicast fashion. Thus, the data delivery between the client and the PointCast server is actually the exact opposite of the view that is presented to the user *in all three dimensions* of the hierarchy of Figure 1. This situation is not unique to PointCast; in fact, it is true for virtually all of the Internet-based push solutions, and stems from the fact that current IP and HTTP protocols do not adequately support push or 1-to-N communication.

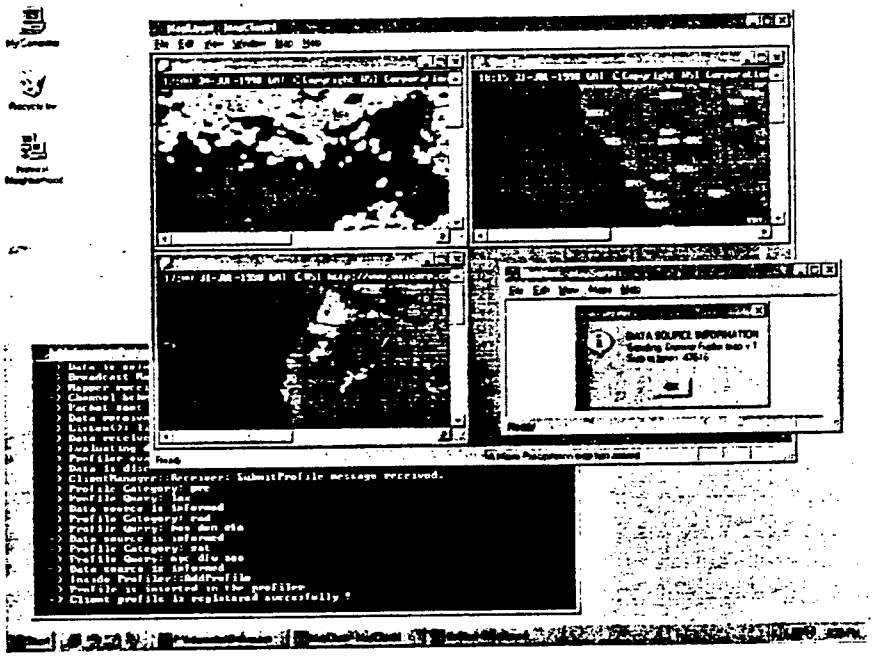


Fig. 2. The Map Dissemination Application

3 An Initial Prototype

As stated in the introduction, our ultimate goal is to build a toolkit of components that can be used to create a DBIS tailored to support a particular set

of dissemination-based applications. In order to better understand the requirements and desired properties of such a toolkit, we have constructed an initial prototype toolkit and have used it to implement a weather map dissemination application.

Figure 2 shows an example screen from this application. In this application one or more "map servers" sends out updated maps of different types (i.e., radar, satellite image, etc.) for different regions of the United States. Clients can subscribe to updates for specific types of maps for specific regions. They can also pose queries to obtain the most recent versions of specific maps. The DBIS components route such queries to the appropriate server(s). In the current prototype, all maps are multicast to all clients — the clients perform additional filtering to avoid displaying unrequested results to the user. In the remainder of this section, we briefly describe the implementation of the prototype toolkit.

3.1 Toolkit Description

Figure 3 shows an example instantiation of a DBIS using the current toolkit. The toolkit consists of four main components. These are shown as lightly-shaded items in the figure. The darker shaded items are software that is not part of the DBIS toolkit, namely, the data sources and clients themselves. The components of the current prototype are:

1. **Data Source (DS) Library** - a wrapper for data sources that encapsulates network communication and provides conversion functions for data.
2. **Client Library** - a wrapper for client programs that encapsulates network communication and provides conversion functions for queries and user profiles. The client library is also responsible for monitoring broadcast and multicast channels and filtering out the data items of local interest that appear on those channels.
3. **Information Broker (IB)** - the main component of the DBIS toolkit. The IB contains communication, buffering, scheduling, and catalog management components and is described in more detail below.
4. **Information Broker Master** - The IB Master is responsible for managing global catalog information about data and about the topology of the DBIS. All IBs must register with the IB Master and all catalog updates must be sent to the IB Master. The presence of the IB Master is one of the major limitations of this initial prototype, as it is obviously a potential scalability bottleneck for the system. A large part of the design effort for the next version of the prototype is aimed at distributing the functions of the IB Master.

3.2 Data Modeling Considerations

The DBIS prototype currently uses a simple data model: the catalog consists of a set of category definitions. Categories are application-specific, that is, each

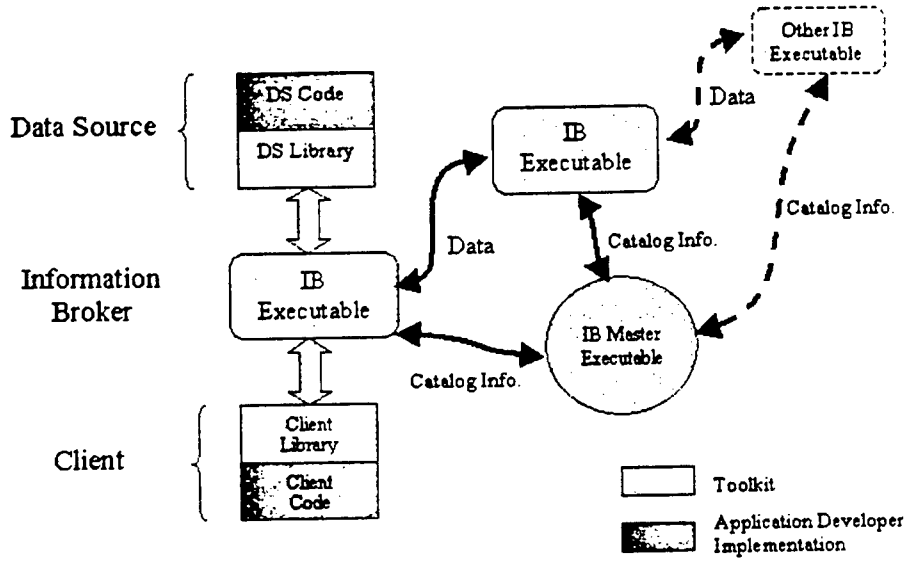


Fig. 3. An Instantiation of a DBIS

application provides its own set of category definitions. Each data item is associated with a single category. In addition, a set of *keywords* can be associated with each data item. Categories and keywords are used in the specification of *queries* and *profiles*. Queries are *pull* requests that are transmitted from a client to a data source. Queries consist of a category and optional keywords. Queries are processed at a data source (or an IB); all data items that match the category (and at least one of the keywords if specified) are sent to the client from which the query originated. In contrast, profiles are used to support *push*-based delivery. When a new data item arrives at an IB, its category and keywords are compared with the user profiles registered at that IB and the item is sent to any clients whose profile indicates an interest in the item. Thus, profiles can be viewed as a form of continually executing query.

Clearly, this simple approach to data modeling must be extended to support more sophisticated applications. We are currently exploring database and WWW-based (e.g., XML) approaches for semi-structured data modeling for use in subsequent versions of the toolkit.

3.3 Information Broker Architecture

As stated above, the Information Broker module contains most of the functionality of the DBIS toolkit. The architecture of an IB is illustrated in Figure 4. Basic components of the IB are the following:

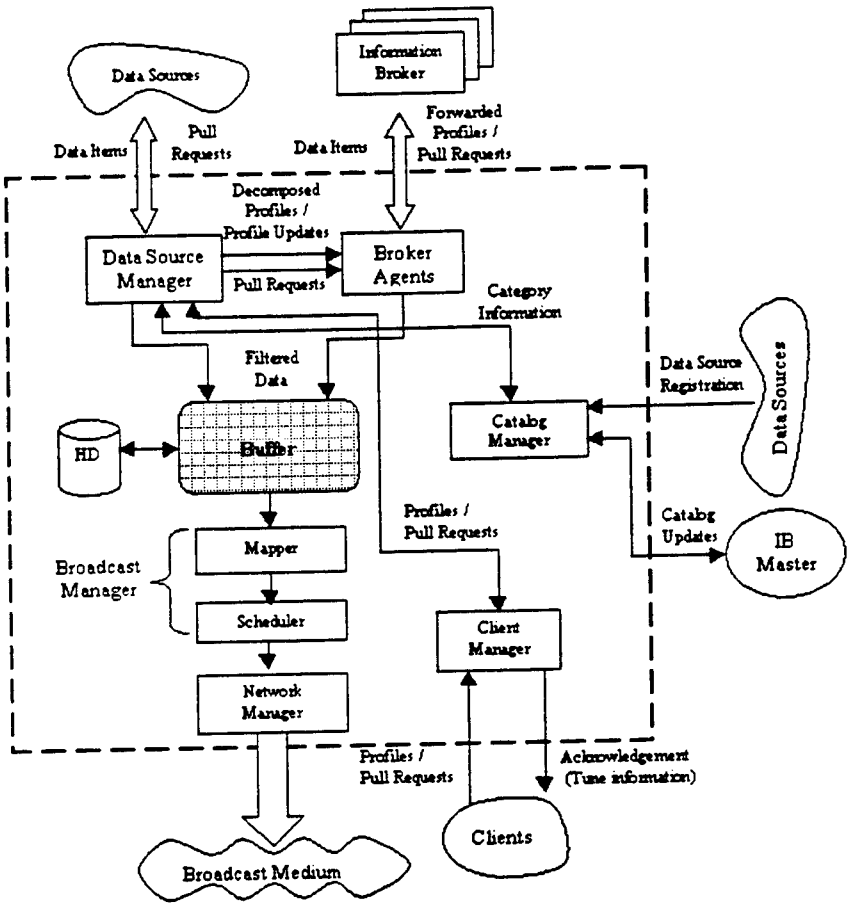


Fig. 4. Information Broker (IB) Architecture

- **Catalog Manager -** This component manages local copies of catalog information for use by the processes running at the broker. Recall that the primary copy of the catalog is managed by the IB Master. All requested changes to the catalog information are sent to the IB Master, which then propagates them to the catalog managers of all other IBs.
- **Data Source Manager -** This component is in charge of receiving and filtering data items obtained from the data sources. It manages a separate listener thread for each data source directly connected to the IB.
- **Broker Agent -** This component is responsible for IB-to-IB interaction, that is, when an IB receives data from another IB rather than directly from a data source.

- **Broadcast Manager** - Once data has been filtered through the data source manager or the broker agent, it is passed to the Broadcast Manager, which has two main components. The *Mapper* assigns the data item to one or more physical communication channels. The *Scheduler* makes decisions about the order in which data items should be placed on those channels.
- **Network Manager** - This is the lowest level of the communication component of the IB. It sends data packets to the network according to the information provided by the broadcast manager.
- **Client Manager** - This module handles all requests that arrive from the clients of the IB. It forwards these requests to the proper modules within the IB and maintains communication sessions with the clients.

4 Example Research Topics

Having described our general approach to building Dissemination-Based Information Systems, we now focus on two examples of the many research issues that arise in the development of such systems.

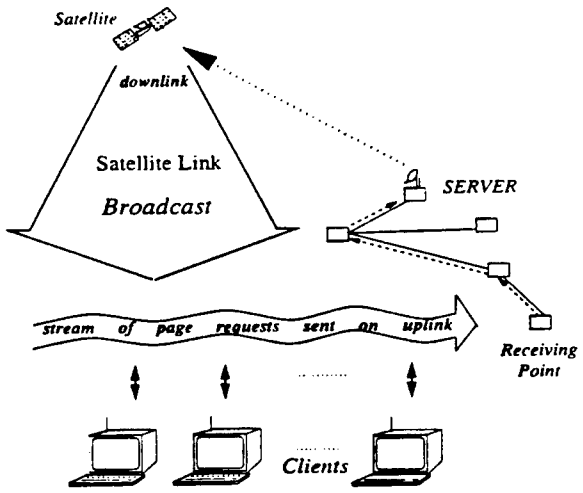


Fig. 5. Example Data Broadcasting Scenario

4.1 Topic 1: On Demand Broadcast Scheduling

As described in Section 2.1, one of the many possible mechanisms for data dissemination uses on-demand (i.e., aperiodic pull) broadcast of data. An example scenario using such data delivery is shown in Figure 5. In this scenario, two independent networks are used: a terrestrial network for sending pull requests to the server, and a "listen only" satellite downlink over which the server broadcasts

data to all of the clients. When a client needs a data item (e.g., a web page or database object) that it cannot find locally, it sends a request for the item to the server. Client requests are queued up (if necessary) at the server upon arrival. The server repeatedly chooses an item from among these requests, broadcasts it over the satellite link, and removes the associated request(s) from the queue. Clients monitor the broadcast and receive the item(s) that they require.

In a large-scale implementation of such a system, an important consideration is the scheduling algorithm that the server uses to choose which request to service from its queue of waiting requests. We have developed a novel on-demand broadcast scheduling algorithm, called RxW [Akso98], which is a practical, low-overhead and scalable approach that provides excellent performance across a range of scenarios.

The intuition behind the RxW scheduling algorithm is to provide a balanced performance for hot (popular) and cold (not so popular) pages. This intuition is based on our observations of previously proposed algorithms. We have observed that two low overhead algorithms, Most Requests First (MRF) and First Come First Served (FCFS) [Dyke86, Wong88], have poor average case performance because they favor the broadcasting of hot or cold pages respectively. A third algorithm, Longest Wait First (LWF) [Dyke86, Wong88] was shown to provide fairer treatment of hot and cold pages, and therefore, good average case performance. LWF, however, suffers from high overhead, making it impractical for a large system.

Based on these observations, we set out to combine the two low-overhead approaches (MRF and FCFS) in a way that would balance their strengths and weaknesses. The RxW algorithm schedules the page with the maximal $R \times W$ value where R is the number of outstanding requests for that page and W is the amount time that the oldest of those requests has been waiting for the page. Thus, RxW schedules a page either because has many outstanding requests or because there is at least one request that has waited for a long time.

The algorithm works by maintaining two sorted lists (one ordered by R values and the other ordered by W values) threaded through the service queue, which has a single entry for any requested page of the database. Maintaining these sorted lists is fairly inexpensive since they only need to be updated when a new request arrives at the server[5]. These two sorted lists are used by a pruning technique in order to avoid an exhaustive search of the service queue to find the maximal $R \times W$ value. This technique is depicted in Figure 6

The search starts with the pages at the top of the R list. The corresponding W value for that page is then used to compute a limit for possible W values. That is, after reading the top page in the R list, it is known that the maximum RxW-valued page cannot have a W value below this limit. Next, the entry for the page at the top of the W list is accessed and used to place a limit on the R value. The algorithm alternates between the two queues and stops when the limit is reached on one of them. This technique prunes the search space while

[5] In contrast, for LWF the ordering can change over time, even in the absence of new requests.

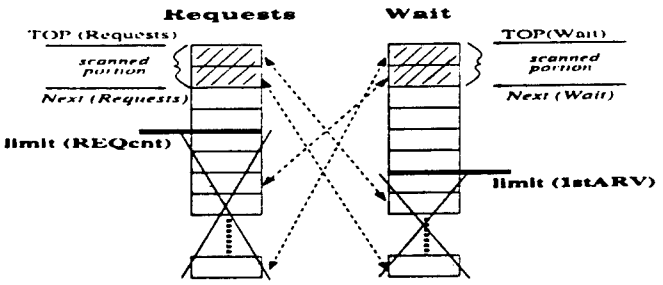


Fig. 6. Pruning the Search Space

still guaranteeing that the search will return the page with the maximum RxW value.

In our experiments [Akso98], the pruning technique was shown to indeed be effective – reducing the number of entries searched by 72%. While such a substantial savings is helpful, it is probably not sufficient to keep the scheduling overhead from ultimately becoming a limiting factor as the system is scaled to the huge applications that will be enabled by the national and global broadcasting systems currently being deployed.

In order to achieve even greater reductions in the search space we developed an approximation-based version of the algorithm. By varying a single parameter α, this algorithm can be tuned from having the same behavior as the RxW algorithm described so far, to being a *constant time* approach. The approximate algorithm selects the *first* page it encounters whose RxW value is greater than or equal to $\alpha \times$ *threshold*, where *threshold* is the running average of the RxW value of the last page that was broadcast and the *threshold* at that time.

The setting of α determines the performance tradeoffs between average waiting time, worst case waiting time, and scheduling overhead. The smaller the value of the parameter, the fewer entries are likely to be scanned. At an extreme value of 0, the algorithm simply compares the top entry from both the R list and the W list and chooses the one with the highest RxW value. In this case, the complexity of making a scheduling decision is reduced to *O(1)*, ensuring that broadcast *scheduling* will not become a bottleneck regardless of the broadcast bandwidth, database size, or workload intensity. We demonstrated the performance, scalability, and robustness of the different RxW variants through an extensive set of performance experiments described in [Akso98].

4.2 Topic 2: Learning User Profiles

User profiles, which encode the data needs and interests of users, are key components of push-based systems. From the user's viewpoint, a profile provides a means of *passively* retrieving relevant information. A user can submit a profile to a push-based system once, and then continuously receive data that are (supposedly) relevant to him or her in a timely fashion without the need for

submitting the same query over and over again. This automatic flow of relevant information helps the user keep pace with the ever-increasing rate of information generation. From the system point of view, profiles fulfill a role similar to that of queries in database or information retrieval systems. In fact, profiles are a form of continuously executing query. In a large publish subscribe system, the storage and access of user profiles can be be resource-intensive. Additionally, given the fact that user interests are changing over time, the profiles must be updated accordingly to reflect up to date information needs.

We have developed an algorithm called *Multi-Modal* (MM), for incrementally constructing and maintaining user profiles for filtering text-based data items [Ceti98]. MM can be tuned to tradeoff effectiveness (i.e., accuracy of the filtered data items), and efficiency of profile management. The algorithm receives relevance feedback information from the users about the documents that they have seen (i.e., a binary indication of whether or not the document was considered useful), and uses this information to improve the current profile. One important aspect of MM is that it represents a user profile as multiple keyword vectors whose size and elements change dynamically based on user feedback.

In fact, it is this *multi-modal* representation of profiles which allows MM to tradeoff effectiveness and efficiency. More specifically, the algorithm can be tuned using a threshold parameter to produce profiles with different sizes. Let us consider the two boundary values of this threshold parameter to illustrate this tradeoff: When the threshold is set to 0, a user profile is represented by a single keyword vector, achieving an extremely low overhead for profile management, but seriously limiting the effectiveness of the profile. At the other extreme, if the threshold is set to 1, we achieve an extremely fine granularity user model, however the profile size equals the number of relevant documents observed by the user, making it impractical to store and maintain profiles. Therefore, it is more desirable to consider intermediate threshold values which will provide an optimal effectiveness/efficiency tradeoff for a given application.

We evaluated the utility of MM by experimentally investigating its ability to categorize pages from the World Wide Web. We used non-interpolated average precision as our primary effectiveness metric and focused on the profile size for quantifying the efficiency of our approach. We demonstrated that we can achieve significantly higher precision values with modest increase in profile sizes. Additionally, we were able to achieve precision values with small profiles that were comparable to, or in some cases even better than those obtained with maximum-sized profiles. The details of the algorithm, experimental setting, and the results are discussed in [Ceti98].

5 Summary

The increasing ability to interconnect computers through internetworking, mobile and wireless networks, and high-bandwidth content delivery to the home, has resulted in a proliferation of dissemination-oriented applications. These applications present new challenges for data management throughout all compo-

nents of a distributed information system. We have proposed the notion of a Dissemination-Based Information System (DBIS) that integrates many different data delivery mechanisms and described some of the unique aspects of such systems. We described our initial prototype of a DBIS Toolkit, which provides a platform for experimenting with different implementations of the DBIS Components. Finally we described our work on two of the many research issues that arise in the design of DBIS architectures.

Data Dissemination and data broadcasting are very fertile and important areas for continued research and development. In fact, we see a migration of data management concerns from the traditional disk-oriented architectures of existing database systems, to the more general notion of *Network Data Management*, in which the movement of data throughout a complex and heterogeneous distributed environment is of paramount concern. Our ongoing research efforts are aimed at better understanding the challenges and tradeoffs that arise in the development of such systems.

References

[Acha95a] S. Acharya. R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", *Proc. ACM SIGMOD Conf.*, San Jose, CA, May, 1995.

[Acha95b] S. Acharya, M. Franklin, S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks", *IEEE Personal Communications*, 2(6), December, 1995.

[Acha97] S. Acharya, M. Franklin. S. Zdonik. "Balancing Push and Pull for Broadcast Data", *Proc. ACM SIGMOD Conf.*, Tucson, AZ, May, 1997.

[Akso98] D. Aksoy, M. Franklin "Scheduling for Large-Scale On-Demand Data Broadcasting" *IEEE INFOCOM '98*, San Francisco, March, 1998.

[Amma85] M. Ammar, J. Wong, "The Design of Teletext Broadcast Cycles", *Perf. Evaluation*, 5 (1985).

[Ceti98] U. Cetintemel, M. Franklin, and C. Giles, "Constructing User Web Access Profiles Incrementally: A Multi-Modal Approach", *In Preparation*, October, 1998.

[Dyke86] H.D. Dykeman, M. Ammar, J.W. Wong, "Scheduling Algorithms for Videotex Systems Under Broadcast Delivery", IEEE International Conference on Communications, Toronto, Canada, 1986.

[Fran97] M. Franklin, S. Zdonik, "A Framework for Scalable Dissemination-Based Systems", Proc. ACM OOPSLA Conference, Atlanta, October, 1997.

[Fran98] M. Franklin, S. Zdonik. "Data in Your Face: Push Technology in Perspective", *Proc. ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 98)*, Seattle, WA, June, 1998, pp 516-519.

[Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communication", *CACM*, 33(2), February, 1990.

[Glan96] D. Glance, "Multicast Support for Data Dissemination in OrbixTalk", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[Herm87] G. Herman, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", *Proc. ACM SIGMOD Conf.*, San Francisco, CA, May, 1987.

[Imie94b] T. Imielinski, S. Viswanathan, B. Badrinath, "Energy Efficient Indexing on Air", *Proc. ACM SIGMOD Conf.*, Minneapolis, MN, May, 1994.

[Oki93] B. Oki, M. Pfluegl, A. Siegel, D. Skeen, "The Information Bus - An Architecture for Extensible Distributed Systems", *Proc. 14th SOSP*, Ashville, NC, December, 1993.

[Rama98] S. Ramakrishnan, V. Dayal, "The PointCast Network" *Proc. ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 98)*, Seattle, WA, June, 1998, p 520.

[Wong88] J. Wong, "Broadcast Delivery", *Proceedings of the IEEE*, 76(12), December, 1988.

[Vish94] S. Viswanathan, "Publishing in Wireless and Wireline Environments", *Ph.D Thesis*, Rutgers Univ. Tech. Report, November, 1994.

[Yan95] T. Yan, H. Garcia-Molina, "SIFT - A Tool for Wide-area Information Dissemination", *Proc. 1995 USENIX Technical Conference*, 1995.

This article was processed using the LaTeX macro package with LLNCS style

Appendix D

DBIS-Toolkit: Adaptive Middleware for Large Scale Data Delivery

Demo description for the ACM SIGMOD Conference, Philadelphia, PA, 1999

DBIS-Toolkit: Adaptable Middleware For Large Scale Data Delivery

Mehmet Altinel, Demet Aksoy, Thomas Baby, Michael Franklin, William Shapiro and Stan Zdonik*

Department of Computer Science
University of Maryland
College Park, MD 20742
{ altinel, demet, thomas, franklin, billshap}@cs.umd.edu

*Department of Computer Science
Brown University
Providence, RI 02912
sbz@cs.brown.edu

Introduction

The proliferation of the Internet and intranets, advances in wireless and satellite networks, and the availability of asymmetric, high-bandwidth links to the home, have fueled the development of a wide range of new ''dissemination-based'' applications. These applications involve the timely distribution of data to a large set of consumers, and include stock and sports tickers, traffic information systems, electronic personalized newspapers, and entertainment delivery. Dissemination-oriented applications have special characteristics that render traditional client-server data management approaches ineffective. These include: tremendous scale, significant overlap in user data needs, and asymmetric data flow from sources to consumers.

The mismatch between the data access characteristics of these applications and the technology used to implement them on the WWW results in scalability problems [Fran98]. For example, WWW based applications employ the HTTP protocol which uses a request-response (or client-server), unicast method of data delivery. Using request-response, each user sends requests for data to the server. The large audience for a popular event can generate huge spikes in the load at servers, resulting in long delays and overloaded servers. Compounding the situation is that users must continually *poll* the server to obtain the most current data, resulting in multiple requests for the same data items from each user. In an application such as an election result server, where the interests of a large part of the population are known *a priori*, most of these requests are unnecessary.

In order to address the needs of this new class of applications, we are developing a Dissemination-Based Information Systems (DBIS) toolkit. The toolkit serves as an adaptable middleware layer that incorporates several different data delivery mechanisms and provides an architecture for deploying them in a networked environment. The toolkit also includes facilities for performance monitoring, which can allow a system developer to examine the impact of using different data delivery mechanisms. We have implemented an initial version of this toolkit and have used it to develop a weather map dissemination application.

DBIS-Toolkit Overview

The DBIS Framework

The basic concepts of the DBIS framework were presented at the OOPSLA 97 conference [Fran97]. A more recent description appears in [Akso98b]. The two major features of the framework are: First, it incorporates a number of different options for data delivery, including traditional request-response, publish/subscribe, Broadcast Disks [Acha95, Acha97] and on-demand broadcast [Akso98a]. Second, it is based on the notion of *network transparency*, which allows different data delivery mechanisms to be mixed-and-matched within a single application. Network transparency is provided through the use of *Information Brokers*, which acquire information and distribute it to other consumers. Brokers are middlemen; a broker acts as a client to some number of data sources, collects and possibly repackages the data it obtains, and then functions as a data source to other nodes of the system. Along the way, brokers may add value to the information, such as integrating it with data from other sources or enhancing its organizational structure. By creating hierarchies of brokers, information delivery can be tailored to the needs of many different users.

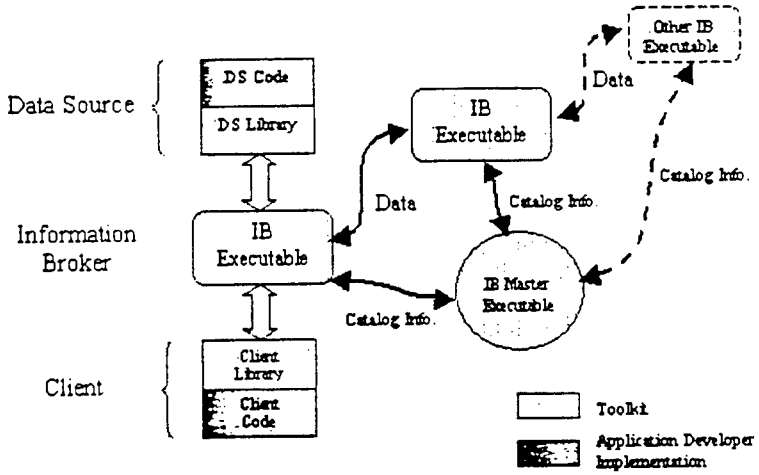


Figure 1: An Instantiation of a DBIS

Toolkit Description

The toolkit provides a set of application programming interfaces (APIs) and libraries that allow a developer to construct and experiment with a DBIS application. Figure 1 shows an example instantiation of a DBIS using the current toolkit. The DBIS-Toolkit consists of four main components (shown as lightly-shaded items in the figure):

Data Source (DS) Library - a data source wrapper that encapsulates network communication and provides conversion functions for data.

Client Library - a client program wrapper that encapsulates network communication and provides conversion functions for queries and user profiles. It also provides monitoring and filtering of broadcast or multicast channels.

Information Broker (IB) - the main component of the DBIS-Toolkit. The IB contains communication,

buffering, scheduling, and catalog management components and is described in more detail below.

Information Broker Master - The IB Master is responsible for managing global catalog information about data and the topology of the DBIS. All IBs must register with the IB Master and all catalog updates must be sent to the IB Master.

In addition to these four components, the toolkit contains a flexible performance monitoring capability that can be used to graphically display real-time performance metrics such as bandwidth and CPU utilization, response times, etc. on a per-IB basis.

Data Modeling

As the focus of this project to date has been on the "plumbing" required to integrate multiple forms of data delivery at the application level, the current prototype uses a very simple data model consisting of categories and keywords within those categories. Categories and keywords are used in the specification of *queries* and *profiles*. Queries are *pull* requests that are transmitted from a client to a data source (via one or more IBs). Queries consist of a category and optional keywords. Queries are ultimately processed at a data source -- all data items that match the category and at least one keyword (if specified) are sent to the client from which the query originated. In contrast, profiles are used to support *push*-based delivery. When a new data item arrives at an IB, its category and keywords are compared with the user profiles registered at that IB and the item is sent to any clients whose profile indicates an interest in the item. Thus, profiles can be viewed as a form of continually executing queries. The integration of more sophisticated data models such as (XML-based) semistructured models, and more flexible IR-style models is one aspect of our on-going development for the toolkit.

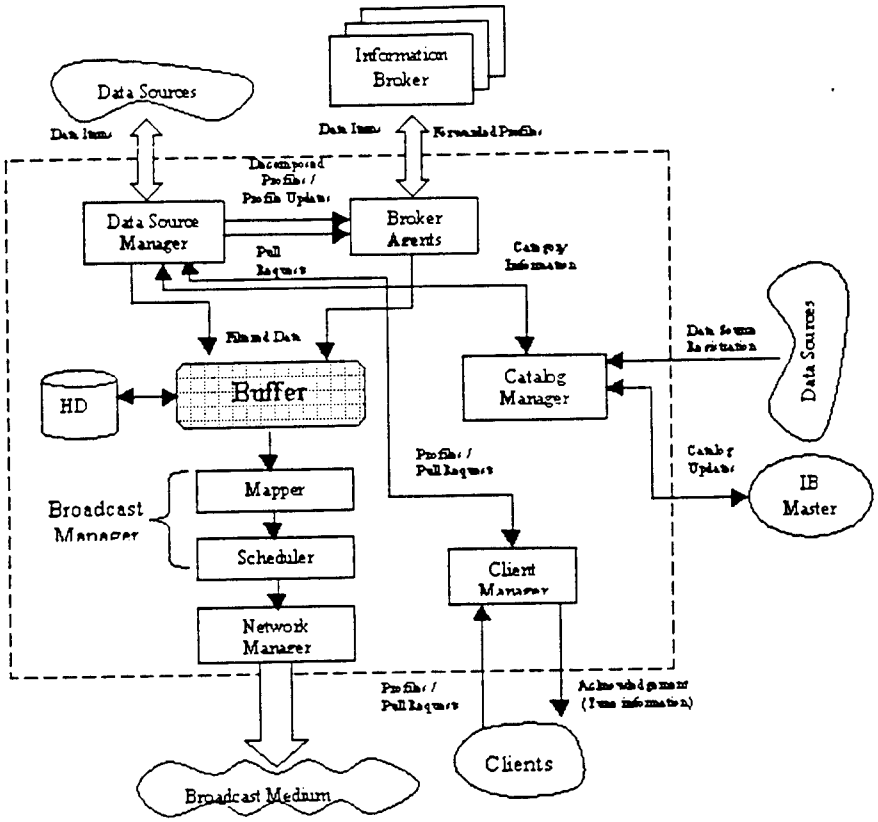


Figure 2: Information Broker (IB) Architecture

Information Broker Architecture

As stated above, the IB contains much of the functionality of the DBIS-Toolkit. The IB module (shown in Figure 2) consists of the following components:

Data Source Manager (DSM) - This component obtains (via push or pull) data items from the data sources and matches them with client pull requests or profiles.

Broker Agent (BA) - This component performs similar functions as the DSM but for sources that are actually other IBs (rather than data sources). In addition, the BA handles other IB-to-IB functions such as profile and request forwarding.

Catalog Manager - This component manages local copies of catalog information for use by the processes running at the broker. All catalog changes are sent to the IB Master, which propagates them to the catalog managers of all other IBs.

Broadcast Manager - Once data have been filtered through the DSM or BA, they are passed to the Broadcast Manager, which has two main components. The *Mapper* assigns data items to one or more physical communication channels. The *Scheduler* makes decisions about the order in which data items should be placed on those channels.

Network Manager - This is the lowest level of the communication component of the IB. It sends data_ packets to the network according to the information provided by the broadcast manager.

Client Manager - This module handles requests that arrive from the IB's clients. It forwards them to the proper modules within the IB and maintains communication sessions with the clients.

A DBIS Application

An initial version of the DBIS-Toolkit has been built using Windows NT and its IP Multicast support. The toolkit has been used to create a weather map dissemination application (see Figure 3). In this application "map servers" send out updated maps of different types (i.e., radar, satellite image, etc.) for different regions of the United States. Clients can subscribe to receive updates for specific types of maps for specific regions. Users can also pose queries to obtain the most recent versions of specific maps or to zoom in on specific regions of the maps. Maps are delivered over unicast or multicast links. The application serves as a demonstration vehicle emphasizing the following unique aspects of the DBIS-Toolkit:

- The incorporation of multiple delivery mechanisms and the ways in which they are supported by the various components of the toolkit.
- The ability to make efficient use of available resources by choosing appropriate delivery mechanisms.
- The exploitation of Network Transparency through the use of multiple levels of Information Brokers.
- The ability to monitor the system dynamically using the graphical performance monitor.

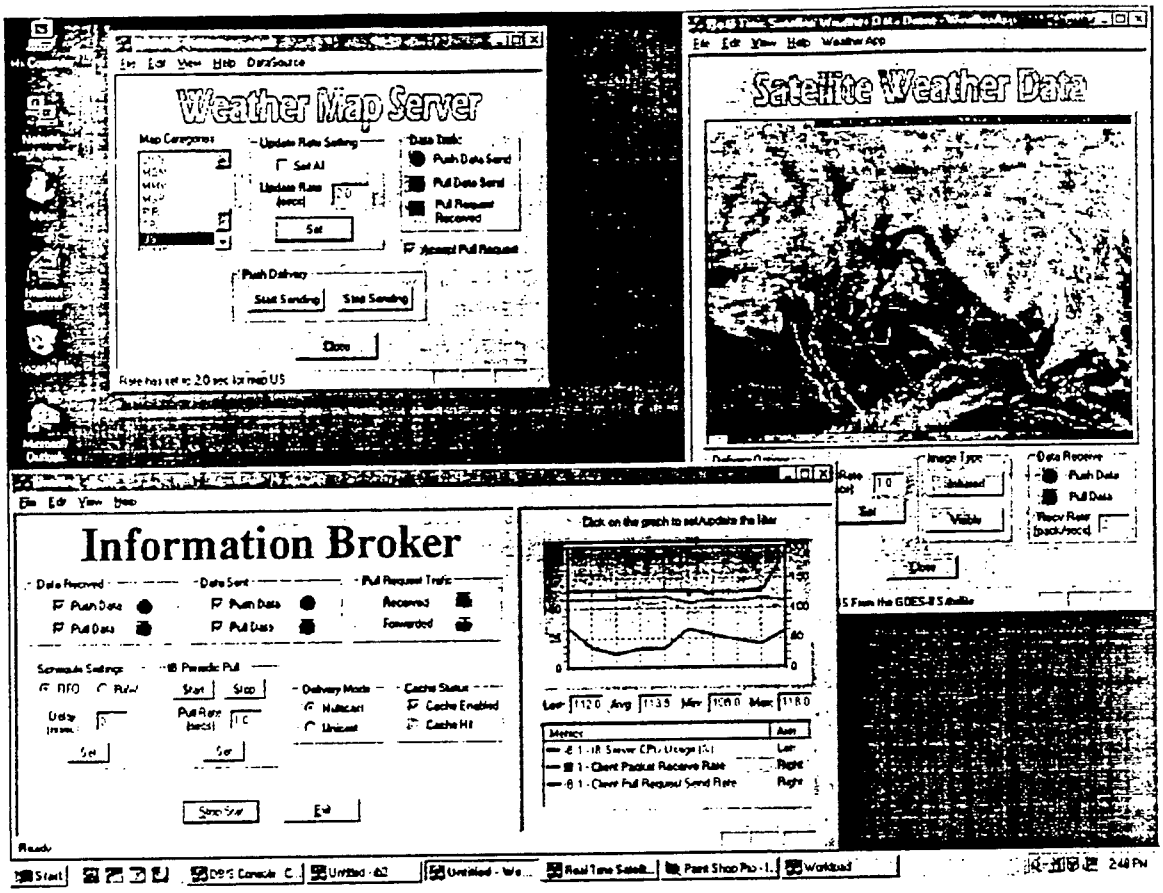


Figure 3: Example DBIS Application

Acknowledgments

The authors would like to thank Rahul Bose, Jane Wang and Rick Rhodes for their contributions to the DBIS prototype design and implementation. This research has been partially supported by Rome Labs agreement number F30602-97-2-0241 under DARPA order number F078, by the NSF under grant IRI-9501353, and by Intel, Microsoft, NEC, and Draper Laboratories.

References

Acha95

S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", *Proc. ACM SIGMOD Conf.*, San Jose, CA, May, 1995.

Acha97

S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Broadcast Data", *Proc. ACM SIGMOD Conf.*, Tucson, AZ, May, 1997.

Akso98a

D. Aksoy, M. Franklin "Scheduling for Large-Scale On-Demand Data Broadcasting" *IEEE INFOCOM '98*, San Francisco, March, 1998.

Akso98b
D. Aksoy, M. Altinel, R. Bose, U. Cetintemel, M. Franklin, J. Wang, S. Zdonik,"Research in Data Broadcast and Dissemination", *Proc. 1st Int'l Conf. on Advanced Multimedia Content Processing*, Osaka University, Osaka, Japan, November, 1998.

Fran97
M. Franklin, S. Zdonik, "A Framework for Scalable Dissemination-Based Systems", Proc. ACM OOPSLA Conference, Atlanta, October, 1997.

Fran98
M. Franklin, S. Zdonik. "Data in Your Face: Push Technology in Perspective", *Proc. ACM SIGMOD Conf.*, Seattle, WA, June, 1998.

Appendix E

A Framework for Scalable Dissemination-Based Systems

A Framework for Scalable Dissemination-Based Systems*

Michael Franklin
University of Maryland
franklin@cs.umd.edu

Stanley Zdonik
Brown University
sbz@cs.brown.edu

Abstract

The dramatic improvements in global interconnectivity due to intranets, extranets, and the Internet has led to an explosion in the number and variety of new data-intensive applications. Along with the proliferation of these new applications have come increased problems of scale. This is demonstrated by frequent delays and service disruptions when accessing networked data sources. Recently, push-based techniques have been proposed as a solution to scalability problems for distributed applications. This paper argues that push indeed has its place, but that it is just one aspect of a much larger design space for distributed information systems. We propose the notion of a Dissemination-Based Information System (DBIS) which integrates a variety of data delivery mechanisms and information broker hierarchies. We discuss the properties of such systems and provide some insight into the architectural imperatives that will influence their design. The DBIS framework can serve as the basis for development of a toolkit for constructing distributed information systems that better match the technology they employ to the characteristics of the applications they are intended to support.

1 Introduction

1.1 The World-Wide Wait

The scenario is all too familiar — a major event, such as a national election, is underway and the latest, up-to-the minute results are being posted on the Web. You want to monitor the results for the important national races and for the races in your state, so you fire up your trusty web browser, point it at the election result web site and wait. and wait, and wait.... What's the problem? It could be any number of technical glitches: a congested network, an overloaded server, or even a crashed server. In a larger sense, however, the problem is one of scalability; the system cannot keep up with the heavy load caused by the (transient) surge in activity that occurs in such situations.

We argue that such scalability problems are the result of a mismatch between the data access characteristics of the application and the technology (in this case, HTTP) used to implement the application. An election result server, such as that of the preceding scenario, is an example of a *data dissemination*-oriented application. Data dissemination involves the delivery of data from one or more *sources* to a large set of *consumers*. Many dissemination-oriented applications have data access characteristics that differ significantly from the traditional notion of client-server applications as embodied in navigational web browsing technology. For example, the election result server has the following characteristics: 1) There is a huge population of users (potentially many millions) who want to access the data; 2) There is a tremendous degree of overlap among the interests of the user population; 3) Users who are following the event closely are interested only in new data and changes to the existing data; and, 4) The amount of data that must be sent to most users is fairly small. When looking at these characteristics, it becomes clear that the request-response (i.e., RPC), unicast (i.e., point-to-point) method of data delivery used by HTTP is the wrong approach for this application.

Using request-response, each user sends requests for data to the server. The large audience for a popular event can generate huge spikes in the load at servers, resulting in long delays and server crashes. Compounding the situation is

*This work has been partially supported by the NSF under grant IRI-9501353, by Rome Labs Agreement Number F30602-97-2-0241 under ARPA order number F078, by an IBM Cooperative Graduate Fellowship, and by research funding and equipment from Intel Corporation.

that users must continually *poll* the server to obtain the most current data, resulting in multiple requests for the same data items from each user. In this example application, where the desires of a large part of the population are known *a priori*, most of these requests are unnecessary.

The use of unicast data delivery likewise causes problems in the opposite direction (from servers to clients). With unicast the server is required to respond individually to each request, often transmitting identical data. For an application with many users, the costs of this repetition in terms of network bandwidth and server cycles can be devastating.

1.2 Is "Push" the Answer?

The above scenario is well-known to web users and, not surprisingly, an increasing number of products are being introduced to address it. A number of these products have received tremendous media attention lately because they are based on a technology called data *Push*. Using data push, the transmission of data to users is initiated without requiring the users to explicitly request it. Examples of systems that employ some form of push technology include Pointcast, Marimba, BackWeb, and AirMedia. Push has also been added to recent versions of the major Web browsers, and the battle for data push standards is well underway.

Systems that are truly implemented with data push can indeed solve some of the scalability problems attributed above to request-response. Since users do not have to poll servers for new and updated data, the number of client requests that must be handled by a server can be reduced dramatically. Simply changing from a client "Pull" model to a push model, however, does not solve all the problems for an application such as the election result server. In particular, performing push to millions of clients using a unicast communication protocol does little to address network bandwidth problems and still requires the server to perform substantial work for each client it is serving. Compounding the confusion is the fact that many systems that provide a "push" interface to users are actually implemented using a programmed polling mechanism. These systems simply save the user from having to click, but do nothing to solve the scalability problems caused by the request-response approach.

The election result server is an example of just one type of dissemination-oriented application. Other examples include news and entertainment delivery, software distribution, traffic information systems, and navigational web browsing. These applications differ widely in the characteristics of the data involved (e.g., size, consistency constraints, etc.), access patterns, and communication channel properties (e.g., symmetric vs. asymmetric, continuously or intermittently connected, etc.). No one data delivery mechanism can provide adequate support for the wide variety of such applications.

To address this need, we are developing a general framework for describing and ultimately constructing Dissemination-Based Information Systems (DBIS). In this framework, push vs. pull is a choice along just one of several dimensions of the design space for data delivery mechanisms. In this paper, we outline a number of data delivery mechanisms and investigate the tradeoffs among them. The goal is to develop a flexible architecture that is capable of supporting a wide range of applications across many varied environments, such as mobile networks, satellite-based systems, and wide-area networks. By combining the various data delivery techniques in a way that matches the characteristics of the application and achieves the most efficient use of the available server and communication resources, the scalability and performance of dissemination-oriented applications can be greatly enhanced.

1.3 Overview of the Approach

We view an integrated DBIS as a distributed system in which the links between the computing elements vary in character: from standard pull-based unicast connections to periodic push over a broadcast channel. A key point is that the character of a link should be of concern only to the nodes on either end. For example, the fact that an information provider receives its data from a broadcast link as opposed to a request-response protocol should make no difference to clients of that provider.

In our approach, we distinguish between three types of nodes: (1) ***data sources*** provide the base data for the application; (2) ***clients*** consume this information; and (3) ***information brokers*** add value to information and redistribute it. By creating hierarchies of these nodes connected by various data delivery mechanisms, the information flow can be tailored to the needs of many different applications.

We aim to provide a *toolkit* of architectural components that can be used to construct a DBIS. A builder of an in-

formation resource would make use of these components to construct the interfaces to their service. Example components include a broadcast generator, a set of dissemination services, a client cache manager, a client prefetcher, a backchannel monitor, etc.

In the remainder of the paper we outline our current ideas on the development of such a toolkit. Section 2 describes several options for data delivery mechanisms (i.e., the "links") and discussesthe tradeoffs among them. Section 3 addresses the various types of nodes in a DBIS. Section 4 uses the DBIS model to describe several existing dissemination-oriented systems. Section 5 outlines issues in the development of a DBIS toolkit. Section 6 lists related work. Finally, Section 7 presents our conclusions.

2 Options for Data Delivery

As stated in the Introduction, a key aspect of the DBIS framework is that it supports a wide variety of links for data delivery between sources and clients. Support for different styles of data delivery allows a DBIS to be optimized for various server, client, network, data, and application properties.

2.1 Three Characteristics

We identify three main characteristics that can be used to compare data delivery mechanisms: (1) push vs. pull; (2) periodic vs. aperiodic; and (3) unicast vs. 1-to-N. Figure 1 shows these characteristics and how several common mechanisms relate to them.

2.1.1 Client Pull vs. Server Push

The first distinction we make among data delivery styles is that of "push vs. pull". Current database servers and object repositories manage data for clients that explicitly request data when they require it. When a request is received at a server, the server locates the information of interest and returns it to the client. This *request-response* style of operation is *pull-based* — the transfer of information from servers to clients is initiated by a client pull. In contrast, push-based data delivery involves sending information to a client population in advance of any specific request. With push-based delivery, the server initiates the transfer.

2.1.2 Aperiodic vs. Periodic

Both push and pull can be performed in either an aperiodic or periodic fashion. Aperiodic delivery is *event-driven* — a data request (for pull) or transmission (for push) is triggered by an event such as a user action (for pull) or data update (for push). In contrast, periodic delivery is performed according to some pre-arranged schedule. This schedule may be fixed, or may be generated with some degree of randomness.[1] An application that sends out stock prices on a regular basis is an example of periodic push, whereas one that sends out stock prices only when they change is an example of aperiodic push.

2.1.3 Unicast vs. 1-to-N

The third characteristic of data delivery mechanisms we identify is whether they are based on unicast or 1-to-N communication. With unicast communication, data items are sent from a data source (e.g., a single server) to one other machine, while 1-to-N communication allows multiple machines to receive the data sent by a data source. Two types of 1-to-N data delivery can be distinguished: multicast and broadcast. With multicast, data is sent to a specific subset of clients. In some systems multicast is implemented by sending a message to a router that maintains the list of recipients. The router reroutes the message to each member of the list. Since the list of recipients is known, it is possible to make multicast reliable; that is, network protocols can be developed that guarantee the eventual delivery of the message to all clients that should receive it. In contrast, broadcasting sends information over a medium on which an unidentified and unbounded set of clients can listen. This differs from multicast in that the clients who may receive the data are not known *a priori*.

2.2 Classification of Delivery Mechanisms

It is possible to classify some existing data delivery mechanisms using the characteristics described above. Such a classification is shown in Figure 1. We discuss several of the leaves in this diagram below.

[1] For the purposes of this discussion, we do not distinguish between fixed and randomized schedules. Such a distinction is important in certain applications. For example, algorithms for conserving energy in mobile environments proposed by Imielinski et al. [Imie94b] depend on a strict schedule to allow mobile clients to "doze" during periods when no data of interest to them will be broadcast.

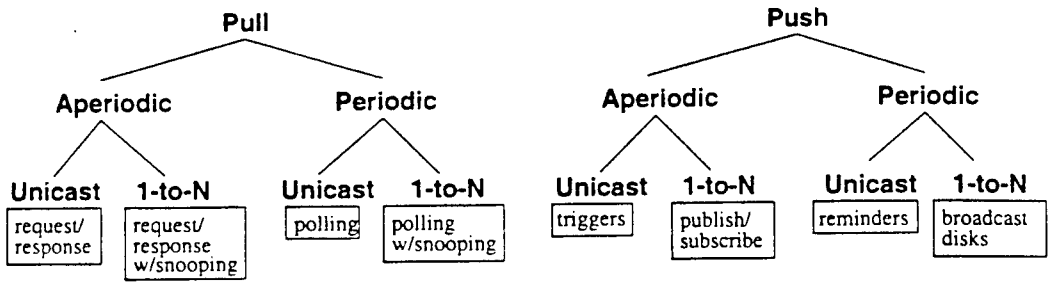


Figure 1: Data Delivery Options

Request/Response - Traditional request/response mechanisms use aperiodic pull over a unicast connection. If instead, a 1-to-N connection is used, then clients can "snoop" on the requests made by other clients, and obtain data that they haven't explicitly asked for.

Polling - In some applications, such as remote sensing, a system may periodically send requests to other sites to obtain status information or to detect changed values. If the information is returned over a 1-to-N link, then as with request/response, other clients can snoop to obtain data items as they go by.

Publish/Subscribe - Publish/subscribe protocols are becoming a popular way to disseminate information in a network [Oki93, Yan95, Glan96]. Publish/subscribe is push-based: data flow is initiated by the data sources, and is aperiodic, as there is no predefined schedule for sending data. Such protocols are typically performed in a 1-to-N fashion, but a similar protocol can be used over a unicast channel, as is done for triggers in active database systems.

Broadcast Disks - Periodic push has been used for data dissemination in many systems such as TeleText [Amma85, Wong88], DataCycle [Herm87, Bowe92], Broadcast Disks [Acha95a, Acha95b] and mobile databases [Imie94a]. Clients needing access to a data item that is pushed periodically can wait until the item appears. As with aperiodic push, periodic push can also be used with both unicast and 1-to-N channels, but we believe that 1-to-N is likely to be much more prevalent.

2.3 Some Example Tradeoffs

As can be seen from the preceding discussion, the design space for data delivery mechanisms is quite large. Choosing the proper mechanism (or combination of them) to use for a given link requires an understanding of the tradeoffs among them. In a recent paper, we studied one such set of tradeoffs; namely, those between broadcasting data using periodic push (Broadcast Disks) and aperiodic pull (request-response with snooping) [Acha97]. Here, we briefly discuss some observations from that study.

The tradeoffs between push and pull in general revolve around the costs of initiating the transfer of data. A pull-based approach requires the use of a backchannel for each request. Furthermore, as described in the Introduction, the server must be interrupted continuously to deal with such requests and has limited flexibility in scheduling the order of data delivery. Also, the information that clients can obtain from a server is limited to that which the clients know to ask for. Thus, new data items or updates to existing data items may go unnoticed at clients unless they periodically poll the server.

Push-based approaches, in contrast, avoid the issues identified for client-pull, but have the problem of deciding which data to send to clients in the absence of specific requests. Clearly, sending irrelevant data to clients is a waste of resources. A more serious problem, however, is that in the absence of requests it is possible that the servers will not deliver the specific data needed by clients in a timely fashion (if ever). Thus, the usefulness of server push is dependent on the ability of a server to accurately predict the needs of clients. One solution to this problem is to allow the clients to provide a *profile* of their interests to the servers. As mentioned above, *Publish/subscribe* protocols are one popular mechanism for providing such profiles.

In [Acha97] we studied a hybrid push/pull broadcast system. In this system, a broadcast server is responsible for allocating a fixed broadcast bandwidth between data items (pages) that are broadcast according to a fixed schedule (i.e., periodic push) and pages that are broadcast in response to

client requests sent over a backchannel (i.e., aperiodic pull). The fundamental performance tradeoff between these two approaches can be seen in in Figure 2. which shows results from [Acha97][2]. The x-axis in the figure models the number of clients (all having identical access rates and distributions) that are accessing data from the broadcast. Thus, at a value of 250. the broadcast is serving 25 times as many clients than at a value of 10. The y-axis indicates the average number of items that a client must watch go by on the broadcast before the item it wants appears.

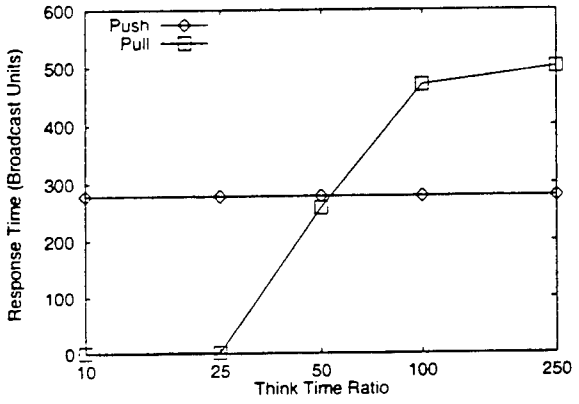


Figure 2: Push vs. Pull for Broadcast

The flat line in the figure (marked by diamonds) indicates the performance of a pure push approach, in which all data is broadcast repeatedly with no requests sent by the clients. This figure was generated using a skewed (Zipfian) access pattern over 1000 items. The broadcast schedule used by the push approach was tailored to support a skewed access pattern through the use of Broadcast Disks which allow the frequency of broadcast for an item to be based on that item's popularity [Acha95a, Acha95b]. As can be seen in the figure, the performance of pure push is *independent* of the number of clients listening to the broadcast here. This is a fundamental property of data broadcast using periodic push — if there is a large overlap in the interests of clients, it provides tremendous scalability in terms of client population.

The other curve in the figure (marked by boxes) shows the performance of a pull-based approach, in which clients submit requests to the server via the backchannel, and the server broadcasts the requested pages in FIFO order.[3] As can be seen in the figure, the pull-based approach exhibits an S-shaped behavior — it provides extremely fast response time for a lightly loaded server, but as the server becomes loaded, its performance degrades, until it ultimately stabilizes (in this case, at a value of 500 items, or half the size of the database being broadcast here).

The behavior of aperiodic pull in this case can be explained as follows. With a lightly loaded system. the server is typically idle so it can respond immediately when a request is received. As the load increases, however, the server saturates and becomes less responsive. Compared to periodic push, it is clear that aperiodic pull demonstrates less scalability in this case. It is, however, important to note that aperiodic pull over a *unicast* channel would be far less scalable — wait time would increase in an unbounded fashion as the server approached saturation. In contrast using broadcast, the performance of aperiodic pull eventually flattens out in this case, because of the overlap in the interests of the client population. Once the server reaches the state where all data items are in the FIFO queue, additional clients receive all of their data by simply "snooping" on the broadcast. In this case the performance of aperiodic pull at saturation is worse than that of periodic push, because the broadcast schedule generated by the FIFO discipline is less well suited to the access pattern than the pre-computed schedule used by periodic push. As discussed in [Acha97], the problems of pull can be exacerbated if the server drops client requests when it becomes overloaded.

The tradeoffs described above give an indication of the kinds of concerns that must be balanced when choosing the proper data delivery mechanism for a given situation. Another set of options arises in the organization of the nodes for a DBIS, as described in the following section.

3 Design Options for Nodes

While the discussion so far has focused on the ways in which data is communicated between computing devices, the nodes in a Dissemination-Based Information System play a crucial role as well: the nodes provide the glue that pastes various data distribution schemes together. A DBIS toolkit should contain classes that model some of the basic features

[2] We briefly summarize these results here, interested readers are referred to [Acha97]for more details

[3] Because a single broadcast of an item satisfies all clients waiting for that item, we do not enqueue a request for an item that is already in the FIFO queue.

of nodes. This section outlines some of those features.

3.1 Classification

In an integrated DBIS, there will be three types of nodes: (1) *data sources*, which provide the base data that is to be disseminated; (2) *clients*, which are net consumers of information; and (3) *information brokers*, that acquire information from other sources, add value to that information (e.g., some additional computation or organizational structure) and then distribute this information to other consumers. By creating hierarchies of brokers, information delivery can be tailored to the needs of many different users.

Information brokers perform many important functions in our architecture. While the previous discussion focused primarily on different modes of data delivery, the brokers provide the glue that binds these modes together. It is typically the expected usage patterns of the brokers that will drive the selection of which mode of delivery to use. For example, a broker that typically is very heavily loaded with requests could be an excellent candidate for a push-based delivery mechanism to its clients.

As we move upstream in the data delivery chain, brokers look like data sources to their clients. Receivers of information cannot detect the details of interconnections any further upstream than their immediate predecessor. This principle of *network transparency* allows data delivery mechanisms to change without having global impact. Suppose that node B is pulling data values from node A on demand. Further, suppose that node C is listening to a cyclic broadcast from node B which includes values that B has pulled from A. Node C will not have to change its data gathering strategy if A begins to push values to B; changes in links are negotiated purely between the two nodes involved.

Of course, nothing is ever simple. In some cases, brokers can also be sources by maintaining their own databases. In this case, the *hybrid* broker can add data of its own to what it receives from its upstream counterparts. The principle of network transparency also protects clients from having to depend on this situation. A data source, be it a pure source, a broker, or a hybrid source, only guarantees that it can provide specific data — independently of where it comes from.

3.2 Caching

While nodes can perform many functions, the most ubiquitous data management facility is caching. Unlike caching in client-server systems, the path from data sources to a client can be of length greater than two. Thus, items might be cached at any of many points along the data path in the network. Thus, caching in this context resembles the kind of proxy caching that one might find in a wide-area network (e.g., the Internet).

While the problems here are very similar to those of any proxy caching scheme, the broad view of data movement available in a DBIS makes the potential solutions much richer. For example, if there are copies of a particular data item in multiple caches, there will always be an issue of how those copies are refreshed when the primary copy is updated. One solution is to send invalidations to each client cache manager. An invalidation message results in the purge of the item from the cache. Alternatively, the new value could be propagated to the client cache managers. For typical client/server systems, invalidation is usually preferable. However, in our broadcast disk studies [Acha96b] we showed that for periodic broadcast, performance can often be improved using propagation.

The decision about how current to keep the cached copies is the same as in other caching mechanisms. Once that has been decided, the means by which it is achieved can vary. In a DBIS, we could propagate (i.e., push) the changes to the clients or wait for the client to request the item again (i.e., pull). In the latter case, if a cache manager cares about keeping items very current, it will have to poll the state of the object often. It is interesting to note that if the data delivery mechanism in a DBIS changes, the means by which updates are propagated (or not) may also need to change.

Deciding which object to evict from the cache when a new candidate arrives is another issue that must be addressed by any cache manager. Many systems use some form of LRU for this purpose. We have shown in previous work [Acha95a] that for some styles of data delivery (e.g., broadcast disks), LRU is not the most effective choice. For cyclic data delivery, in which different items can have different arrival frequencies, a cost-based caching scheme performs significantly better.

In a DBIS, the modes of data delivery might change. In such an environment, the caching policy could change

to match the prevailing conditions. We will need heuristics for deciding the appropriate caching policies for a particular configuration of distributed components. As an example, if node B initially pulls data from node A, B might reasonably use LRU as its caching policy. When A creates a broadcast disk which is read by B, B might then change its caching policy to a cost based scheme similar to the one that we propose in [Acha95a].

3.3 Value-Added Nodes

Some nodes may also add value to data as it passes through, by performing specific computations on that data. The computations can be simple or complex, or they can act on single values or sets of values. Other nodes may simply pass values on to other nodes.

As an example, suppose node A pushes stock prices for Fortune 500 companies that are picked up by node B. Node B keeps a database of previous stock prices and when a new price for the day is picked up from node A, it calculates the difference between the most current price and yesterday's close, and pushes this value out to yet another community. Node B is a push-based, value-added server. Of course, it need not be based on push. Other clients could pull stock deviations from B as well.

Another kind of value-added service that a node can perform is *merging* of values from multiple sources. Merging can occur in several ways. The first involves multiple sources that maintain similar information. The merge node can make the most reliable or most current version of a value available. Alternatively, multiple sources may maintain a set of values which the merge node combines to a single value. An example of this might involve nodes that maintain demographic information for towns including their current population. Another node may read these values and consolidate them into a single population figure for the state.

Nodes can also perform the service of *filtering*. A filtering node will receive a large volume of data from another node, only some fraction of which it makes available to its clients. For example, a node could receive all stock prices from the NYSE and provide information about only the Fortune 500 stocks to its clients.

3.4 Recoverable Nodes

Often it will be useful to make guarantees about the reliability of some node. Thus, nodes that implement some degree of recoverability will be a useful component in a DBIS. Consider a node that must guarantee the delivery of the latest version of IBM's stock price. Such a node must not lose its information in the event of a failure. That is, if the information was received, then the node must be able to guarantee that it will eventually be made available to its clients.

Of course, having recoverable brokers is not enough on its own to guarantee that nodes will not miss disseminated information while they are down. In order to address this issue, a scheme like reliable multicasting would have to be used. Reliable multicasting will eventually deliver all messages, but it cannot make real-time guarantees about when an object will arrive.

3.5 The Burden of Push

As mentioned in Section 2.3, any node that provides a push service must do so on the basis of some knowledge of the access patterns of its client base. If the node pushes data that few clients care about, then bandwidth is wasted. The trick is to broadcast items that are of interest to a large segment of the user community. This, of course, is only possible if there is high commonality of interest for at least some data items.

In order to optimize its push schedule, the server must rely on profiles of user needs. Profiles could be learned by servers if clients provide feedback about the effectiveness of the push schedule. Alternatively, a client could communicate a profile to the server at appropriate times, such as when it begins to listen to the push, at regularly scheduled intervals, or whenever the client notices that the current schedule deviates significantly from what it would like to see.

What would such a profile look like? A profile is very much like a continuously executing query [Terr92]. In other words, it is a predicate that indicates the items that the client would like to see. It is continuously executing because the server will push items as long as there are currently valid profiles that match the items.

Profiles can be interpreted to mean that whenever a new item is added to the database that matches a profile, the owner of that profile will receive the new data. On the

other hand, the profile could be treated more as a hint to the server indicating interest with no requirement on the server's part to send matching items. In this case, the server may choose to conserve bandwidth and not send a matching item in order to best serve the client community as a whole.

4 Systems Viewed as DBIS

In this section, we describe some existing systems using the concepts of our DBIS framework.

4.1 Pointcast

Pointcast is a dissemination service that has attracted a large population of users. It obtains profiles from users that describe their interests, and then uses these profiles to assemble and update customized "newspapers" from a database of current stories.

The Pointcast system has been touted as one of the first push-based systems. This is not exactly true. Other systems such as Teletex [Amma85], BCS at MIT [Giff90], and Datacycle [Herm87] used push long before Pointcast. However, Pointcast was one of the first push-based systems to achieve wide-spread use. It is instructive, therefore, to see exactly how push is used in Pointcast 1.0 [4].

From the point of view of a DBIS, the use of push within Pointcast is extremely limited. In fact, in terms of the network architecture, push is non-existent; that is, the flow of requests and responses within the global architecture is pull-based. The Pointcast client on a user's workstation generates requests for news stories that match the user's profile. For example, if the user indicates an interest in the computer industry, the Pointcast client polls the Pointcast server for news stories with the keyword "computer industry" whenever the Pointcast screen saver is enabled. All of these requests can generate lots of network traffic.

So, where's the push? If we look at Figure 3, we see that there are essentially two processes in the client machine. One of these processes is responsible for *pulling* the latest news stories down to the user's machine, and the other is responsible for displaying these stories on the user's screen. The push really occurs between these two components. When the pull-based story acquisition module gets a new story, it pushes it to the screen manager. From the user's point of view, this *is* push because things are happening to the screen without any intervention. The use of push as a technique for managing heavy network loads, however, is not part of the design.

[4] Hereafter, referred to as Pointcast.

4.2 Broadcast Disks

Our own work on broadcast disks is based on a model of data delivery that is virtually the direct opposite of that described above for Pointcast (see Figure 4).

In our model, an application process on the client workstation behaves exactly as it would in a traditional pull-based environment. It generates pull requests as it needs data and blocks until that data is received.

The server, however, proactively sends data to the client community in advance of any request (i.e., push). A process on the client listens to the broadcast stream and picks up data items for which the application might be waiting. Thus, the places where pushes and pulls happen have been inverted over the Pointcast case.

It should be noted that in the broadcast disk case, the push is periodic and is scheduled by the server. In the Pointcast case, the pull is also periodic, but the interval is set by the user.

4.3 SIFT

The SIFT [Yan95] system was developed at Stanford University as a way to disseminate documents to a user community. SIFT combines data management ideas from information retrieval with a *publish/subscribe* model for dissemination. We describe the way the publish/subscribe model works in terms of our DBIS architecture.

Looking at Figure 5, we see three active components: the document source, the SIFT server, and a SIFT client (one of potentially many). The connection between the document source and the SIFT server (on the left side of the figure) is push-based, unicast, and aperiodic. The document source could alternatively deliver new documents through a 1-to-n broadcast medium, such as a satellite feed, if there were multiple interested recipients (SIFT servers or otherwise). A backchannel (not shown in the figure), is used only to set up the initial connection. Thereafter, the document source forwards all new documents to the SIFT server. There is no filtering that happens on this link. We could think of the profile held at the document source for the SIFT server as

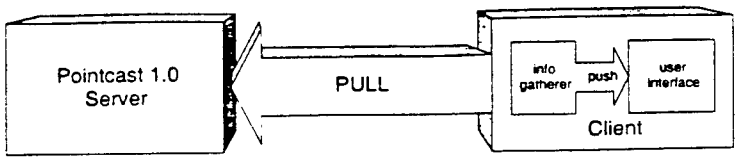


Figure 3: Pointcast 1.0

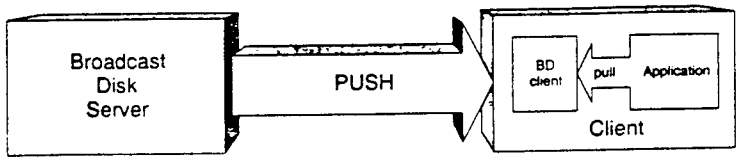


Figure 4: Broadcast Disks

being *send everything*.

The connection between the SIFT server and a given SIFT client (shown on the right side of the figure) is also push-based, unicast, and aperiodic. In this case, though, the client profile that is held at the SIFT server is customized for each client. It consists of a series of keywords and weights that describe documents of interest to that client. The SIFT server provides novel technology for indexing client profiles. Such an index is used for matching profiles against newly arriving documents. This indexing technique allows the server to accommodate a large client population with reasonable performance. The original SIFT prototype disseminated entire articles to clients. With the existence of the web, it becomes possible to send short article descriptions plus the corresponding URLs to conserve bandwidth.

It should be noted that clients get exactly what their profiles specify and nothing more. This is in contrast to a 1-to-n (broadcast) style of delivery in which all clients see the same information stream. It is the server's responsibility to optimize this stream to suit the needs of the largest number of users. It is unlikely that such a stream will be optimal for any one user.

5 Putting it All Together

In the preceding discussion, we described a vision of how distributed information systems should be built in the future. Our framework focused on techniques for delivering data in wide-area network settings in which nodes and links reflect extreme variation in their operating parameters. By adjusting the delivery mechanism to match these characteristics, we believe that we can achieve high performance and scalability without the need to invest in additional hardware. In this section, we briefly discuss our approach to this problem and outline some of the open research questions.

5.1 Toolkit Approach

We intend to realize our solutions to the problems of designing a DBIS through a toolkit that provides the proper components from which any DBIS could be built. This toolkit can be thought of as a set of object classes that support concepts such as *network connections* and *local caches*.

A key part of the toolkit will be a set of classes to allow distributed nodes to negotiate in order to establish a proper connection. This is required at several levels. At the highest level, the nodes must agree on how data is to be transferred. A client node that is relying on data from some server must know whether that server will be using push or accepting requests. There are also handshaking protocols that must occur at lower levels. For example, if a push-based broadcast connection is to be established in an Ethernet, the nodes must agree on which Ethernet address will be used for that broadcast. The parties must also agree on the parameters that will be used to configure that broadcast. For example, if it is a broadcast disk, the frequency of broadcast of each item is of interest to the clients.

The usefulness of a toolkit will rely on the precise definition of the DBIS classes. These classes must be of general utility. Also, as indicated in Section 2.3, the definition of

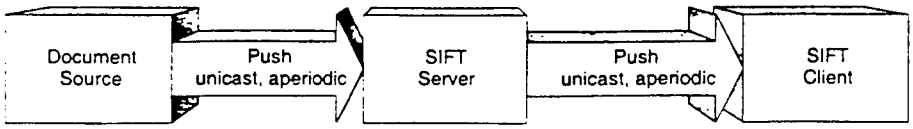


Figure 5: High-Level SIFT Architecture

these classes must be based on a substantial body of experimental results that help to delineate the sometimes subtle tradeoffs.

5.2 Dynamic Reconfiguration

A network can be characterized by prevailing loads on the nodes and the connections. This characterization changes rapidly, and a responsive DBIS must be able to adapt to these changes. Thus, our vision of a fully functional DBIS includes facilities to support the dynamic reconfiguration of the data delivery mechanisms.

A key element of a reconfiguration facility is a statistics gathering component that collects the right performance numbers and that can intelligently select among the available delivery options. This is not a simple matter. Our previous experiments in the area of broadcast disks has shown that the design space here is very complex with many places in which intuitions from more traditional distributed system design often produces poor results.

5.3 Some Design Issues

In addition to the plumbing issues that we have discussed so far, there are some higher-level issues that must be addressed in developing an integrated DBIS. In the following, we briefly outline some of these issues:

- *Bandwidth Allocation* - For a given link, policies are needed for allocating bandwidth among the various data delivery mechanisms.
- *Push Scheduling* - For the push-based approaches, intelligent scheduling is necessary in order to obtain the maximal benefit from the available bandwidth. Scheduling must also take into account the likelihood and distribution of transmission errors. Also, for periodic push, the broadcast should include index and/or schedule information that describes the objects that are to appear in the upcoming broadcast. Such information allows clients to minimize the amount of time and/or processing they devote to monitoring the broadcast and can aid in storage management decisions.
- *Client Storage Management* - Clients must allocate their storage resources among the data obtained through the various delivery mechanisms. Furthermore, as stated earlier, different methods of data delivery impose differing demands on the policies for client caching and prefetching. Furthermore, in some cases (e.g., mobility), storage management must also take into account the likelihood of disconnection and of data becoming stale due to updates or expiration.
- *User Profiles and Feedback* - Profiles of client needs are key for making allocation, scheduling and other policy decisions at both clients and servers. The form of the profiles will be important to achieve the most effective use of the medium. For example, access probabilities are one specific representation of the client needs. The server must also have effective models for combining client profiles. The integration of a *backchannel* from clients to servers is needed to allow for updating profiles and making additional requests.
- *Security Issues* - Another set of important issues that must be addressed revolves around the security and privacy concerns that arise in any distributed information system. The emphasis on one-to-N communication in a DBIS, however, increases the significance of such issues.
- *Consistency Issues* - The final issue we list here is the maintenance of data consistency, particularly in the face of possibly intermittent connection. Two types of consistency must be considered. First, guarantees on the timeliness of individual data items must be provided if required by the clients. Second, *mutual consistency* across multiple items will be required in some

instances. All types of consistency must be provided in a flexible manner, so that tradeoffs between consistency and responsiveness can be made on a case-by-case basis.

6 Related Work

Work on distributed object computing has generated many important standards and systems. CORBA [OMG91] and DCE [OSF94], for example, are two important approaches to system interoperability. This work is not incompatible with the notion of a DBIS. A DBIS can be thought of as infrastructure for such object-oriented middleware.

There is much previous work that relates to the architectural issues of a DBIS. The brief discussion that follows samples some of the work that is most related to the issues presented in this paper.

The management of data in distributed settings has a long history. The preponderance of previous work assumes that data is requested when needed (i.e., pull) and that servers respond to these requests in an orderly fashion. Some of this work has occurred in a client/server database setting [Fran96a] while other work has been done in the distributed file system context [Levy90]. There has been a lot of work on caching in these environments, much of which has focused on the maintenance of cache consistency in the face of updates.

More recently, there has been work on data management issues for wireless environments [Katz94]. Some of work in this area has focused on satellite-based systems [Dao96, Dire96] in which the downstream bandwidth is quite high.

The idea of the publish/subscribe model as a dissemination mechanism has been used in many contexts including SIFT [Yan95] and the Information Bus[Oki93].

There has also been work on broadcasting in Teletex systems [Amma85, Wong88]. [Wong88] presents an overview of some of the analytical studies on one-way, two-way and hybrid broadcast in this framework.

The Datacycle Project [Bowe92, Herm87] at Bellcore investigated the notion of using a repetitive broadcast medium for database storage and query processing. An early effort in information broadcasting, the Boston Community Information System (BCIS) is described in [Giff90]. BCIS broadcast news articles and information over an FM channel to clients with personal computers specially equipped with radio receivers. Both Datacycle and BCIS used a flat broadcast (i.e., all items have the same frequency). The mobility group at Rutgers [Imie94a, Imie94b] has done significant work on data broadcasting in mobile environments. A main focus of their work has been to investigate novel ways of indexing in order to reduce power consumption at the mobile clients. Some recent applications of dissemination-based systems include information dissemination on the Internet [Yan95, Best96], and Advanced Traveler Information Systems [Shek96].

Our work on Broadcast Disks differs from these in that we consider multi-level disks and their relationship to cache management. In [Acha95a], we proposed an algorithm to generate Broadcast Disk programs and demonstrated the need for cost-based caching in this environment. Recently, [Baru96] gave an algorithm to determine the parameters controlling a broadcast program. In [Acha96a], we showed how opportunistic prefetching by the client can significantly improve performance over demand-driven caching. More recently, in [Acha96b], we studied the influence of volatile data on client performance and showed that the Broadcast Disk environment can be made very robust in the presence of updates. In [Acha97], we explored the tradeoff between cyclic broadcast and pull.

7 Conclusions

The increasing ability to interconnect computers through internetworking, mobile and wireless networks, and high-bandwidth content delivery to the home, has resulted in a proliferation of dissemination-oriented applications. A key attribute of many such applications is their huge scale. These applications present new challenges for data management throughout all components of a distributed information system. We have proposed the notion of a dissemination-based information system that integrates many different data delivery mechanisms and types of information brokers. We described some of the unique aspects of such systems and discussed how several existing dissemination-based architectures fit in to the DBIS model.

The ideas presented in this paper have grown out of our previous work on the Broadcast Disks paradigm for data delivery. A key lesson from that work was the importance of applying a data management perspective to distributed systems architecture issues. We are currently completing

a prototype that combines the push-based Broadcast Disks with a pull-based broadcast model. We view that prototype as the first step in the development of a generic DBIS toolkit that will support the creation of a variety of large-scale dissemination-based applications across several different communication media.

Acknowledgments

We would like to thank Swarup Acharya for his contributions to these ideas through the development of the Broadcast Disks paradigm and Demet Aksoy who has provided us with important insights into the properties of broadcast scheduling.

References

[Acha95a] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", *Proc. ACM SIGMOD Conf.*, San Jose, CA, May, 1995.

[Acha95b] S. Acharya, M. Franklin, S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks", *IEEE Personal Communications*, 2(6), December, 1995.

[Acha96a] S. Acharya, M. Franklin, S. Zdonik, "Prefetching from a Broadcast Disk", *12th International Conference on Data Engineering*, New Orleans, LA, February, 1996.

[Acha96b] S. Acharya, M. Franklin, S. Zdonik, "Disseminating Updates on Broadcast Disks", *Proc. 22nd VLDB Conf.*, Bombay, India, September, 1996.

[Acha97] S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Data Broadcast", *Proc. ACM SIGMOD Conf.*, Tucson, AZ, May, 1997.

[Amma85] M. Ammar, J. Wong, "The Design of Teletext Broadcast Cycles", *Perf. Evaluation*, 5 (1985).

[Baru96] S. Baruah and A. Bestavros, "Pinwheel Scheduling for Fault-tolerant Broadcast Disks in Real-time Database Systems", Technical Report TR-96-023, Boston University, August, 1996.

[Best96] A. Bestavros, C. Cunha, "Server-initiated Document Dissemination for the WWW", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[Bowe92] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, A. Weinrib, "The Datacycle Architecture", *CACM*, 35(12), December, 1992.

[Care91] M. Carey, M. Franklin, M. Livny, E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures", *Proc. ACM SIGMOD Conf.*, Denver, June, 1991.

[Dao96] S. Dao, B. Perry, "Information Dissemination in Hybrid Satellite/Terrestrial Networks", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[Dire96] Hughes Network Systems, DirecPC Home Page. `http://www.direcpc.com/`, Oct, 1996.

[Erik94] H. Erikson, "MBONE: The Multicast Backbone", *CACM*, 37(8), August, 1994.

[Fran96a] M. Franklin, *Client Data Caching: A Foundation for High Performance Object Database Systems*, Kluwer Academic Publishers, Boston, MA, February, 1996.

[Fran96b] M. Franklin, S. Zdonik, "Dissemination-Based Information Systems", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communication", *CACM*, 33(2), February, 1990.

[Glan96] D. Glance, "Multicast Support for Data Dissemination in OrbixTalk", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[Herm87] G. Herman, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", *Proc. ACM SIGMOD Conf.*, San Francisco, CA, May, 1987.

[Imie94a] T. Imielinski, B. Badrinath, "Mobile Wireless Computing: Challenges in Data Management", *CACM*, 37(10), October, 1994.

[Imie94b] T. Imielinski, S. Viswanathan, B. Badrinath, "Energy Efficient Indexing on Air", *Proc. ACM SIGMOD Conf.*, Minneapolis, MN, May, 1994.

[Katz94] R. Katz, "Adaption and Mobility in Wireless Information Systems", *IEEE Personnal Comm.*, 1st Quarter, 1994.

[Levy90] Levy, E., Silbershatz, A., "Distributed File Systems: Concepts and Examples", *ACM Computing Surveys*, 22(4), December, 1990.

[OMG91] Object Management Group and X/Open, "Common Object Request Broker: Architecture and Specification", *Reference OMG 91.12.1*, 1991.

[Oki93] B. Oki, M. Pfluegl, A. Siegel, D. Skeen, "The Information Bus - An Architecture for Extensible Distributed Systems", *Proc. 14th SOSP*, Ashville, NC, December, 1993.

[OSF94] Open Software Foundation, "Introduction to OSF DCE", *Prentice Hall*, Englewood Cliffs, NJ, 1994.

[Shek96] S. Shekhar, A. Fetterer, D. Liu, "Genesis: An Approach to Data Dissemination in Advanced Traveller Information Systems", *IEEE Data Engineering Bulletin*, 19(3), September, 1996.

[Terr92] D. Terry, D. Goldberg, D. Nichols, "Continuous Queries Over Append-Only Databases", *Proc. ACM SIGMOD Conf.*, San Diego, CA, June, 1992.

[Wong88] J. Wong, "Broadcast Delivery", *Proceedings of the IEEE*, 76(12), December, 1988.

[Yan95] T. Yan, H. Garcia-Molina, "SIFT - A Tool for Wide-area Information Dissemination", *Proc. 1995 USENIX Technical Conference*, 1995.

[Zdon94] S. Zdonik, M. Franklin, R. Alonso, S. Acharya, "Are 'Disks in the Air' Just Pie in the Sky?", *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December, 1994.

Appendix F

Data in Your Face: Push Technology in Perspective

"Data In Your Face": Push Technology in Perspective*

Michael Franklin
University of Maryland
franklin@cs.umd.edu

Stan Zdonik
Brown University
sbz@cs.brown.edu

Abstract

Push technology has recently generated a tremendous amount of media attention, commercial activity, and controversy. The wide range of opinions on push is understandable given that it represents a major departure from the way distributed information systems have traditionally been built. Adding to the noise, however, is confusion about the basic principles of push and where it fits in to the world of data delivery. For example, many discussions on the topic blur the distinction between push and broadcast. We argue that this confusion stems from two fundamental causes: First, push is just one dimension of a larger design space of data delivery mechanisms. Second, networked information systems can employ different data delivery options between different sets of information producers and consumers. In this short paper we characterize the design space for dissemination-based information systems and applications, and show how current "push" solutions fit into this space. We then use this framework highlight how the implementation of current Internet-based push solutions differs from the appearance that they present to users.

1 Introduction

Push technology stems from a very simple idea. Rather than requiring users to explicitly request (i.e., "pull") the information that they need, data can be sent to users without having them specifically ask for it. The advantages of push are straightforward. The traditional pull approach requires that users know *a priori* where and when to look for data or that they spend an inordinate amount of time polling known sites for updates and/or hunting on the network for relevant sites. Push relieves the user of these burdens. The problems of push are also fairly obvious. Push transfers control from the users to the data providers, raising the potential that users receive irrelevant data while not receiving the information they need. These potential problems can arise due to issues ranging from poor prediction of user interests to outright abuse of the mechanism, such as "spamming". The "in-your-face" nature of push technology is the root of both its potential benefits and disadvantages.

Push technology has been around in various forms for as long as people have been communicating. Examples range from newspapers, to telephones, to radio and television, to E-mail. Early work on using computer networks for pushing data was performed in the 1980's. The Boston Community Information System at MIT [Giff90], Teletext systems for distributing data over broadcast media [Amma85, Wong88], and the Datacycle database machine [Herm87], are all examples of systems that incorporated some form of push technology. Recently, however, the combination of push technology with the Internet and Web (sometimes referred to as *Webcasting*) has generated a ground swell of excitement, commercial activity, and controversy.

1.1 The Push Phenomenon

In February 1996, PointCast made its client software available for free downloading over the Internet, setting off a wave of interest in push technology. The idea was appealing: rather than using your idle desktop machine as a display ground for flying toasters, PointCast would turn it into an active information terminal that would display headlines, weather forecasts, stock prices, sports scores, etc., with the appearance of having real-time updates. By specifying a *profile*, users could indicate their interests to the system, and the display would be tailored to these interests.

For anyone who tried the software, the reaction was immediate; this represented a paradigm shift in the way one could think about using the Internet as an information delivery tool. Push technology on the Internet represented a new and untapped medium. The computer trade press became in-

*This work has been partially supported by Rome Labs agreement number F30602-97-2-0241 under DARPA order number F078, by the NSF under grant IRI-9501353, and by research grants from Intel and NEC.

To appear in the ACM SIGMOD International Conference on the Management of Data, Seattle, WA, June, 1998.

undated with articles about push technology and dozens of companies touting push-based solutions arrived on the scene. A new jargon of data delivery was developed, with terminology borrowed from broadcast media. Users of push technology could *tune* into *channels* that contained *broadcasts* of information on particular topics.

By the end of 1996, the excitement had spilled over into the mainstream press. A steady stream of articles about push technology appeared in venues such as the *New York Times* and the *Wall Street Journal*.[1] In February 1997, *Business Week* magazine published a Special Report section entitled "A Way Out of the Web Maze", which argued that Webcasting could solve many of the Web's problems, such as information overload and the inability for users to find the data they need. Similar sentiments were echoed by numerous vendors and technology pundits.

The peak of the media hype for push technology was reached in March of 1997 when the cover article of *Wired* magazine blared: "Push! Kiss your browser goodbye". This article began by declaring: "Remember the browser war between Netscape and Microsoft? Well forget it. The Web browser itself is about to croak. And good riddance.". While the article was certainly provocative and clearly overstated, the argument it made was simply that push technology would change the Web from a passive library of information into a networked, immersive medium for information and entertainment delivery. Despite this simple message, the article seemed to epitomize both the promise of push technology and the potential for overselling its virtues.

1.2 The Inevitable Backlash

Around the time of the *Wired* article, the voices of dissent began to make themselves heard. A March 1997 *New York Times* CyberTimes article by James Gleick stated: "... the promotion of Push is the silliest piece of puffery to waft along in several seasons. ... The failure of Push is preordained.". A July 1997 article in the on-line net-zine *webmonkey* (published by the same company that publishes *Wired*), was entitled simply "Why Channels Suck". A somewhat more technical article at the CNET on-line site entitled "Networks Strained By Push", described a study indicating that push technologies were using an inordinate portion of corporate network bandwidth. Finally, a *Byte* magazine article in August 1997 had the tag line: "Web push technology is exploding — even though there's no such thing.". The *Byte* article went on to explain (correctly) that current push technology is "really pull++".

[1] Many of these articles had titles such as "When Push Comes to Shove", "The Pull of Push", or "X Gets Pushy" (where X is some product or company). The observant reader will notice that we have resisted such temptations for this paper.

1.3 The Current Situation

Recently, the media turmoil over push has settled down and expectations for the technology (at least for the short term) have lowered to arguably more reasonable levels. Still, the commercial activity in the area is impressive. As of January 1998, a register of push technology vendors listed 49 companies with announced products (see David Strom's site at http://www.strom.com/imc/t4a.html). Many other companies who have not yet announced products are working on push-based solutions. The major web browser vendors, Netscape and Microsoft, have both incorporated push into their products.

A development indicating a degree of maturation of the field is Microsoft's proposal of the Channel Definition Format (CDF) standard to the World Wide Web Consortium (W3C). CDF is a language that web publishers can use to turn their content into "Channels" that can be exploited by push (or "pull++") technologies. CDF allows the specification of metadata about a website, including a searchable title and abstract and information about the structure and update schedule of the site. A number of the major push vendors such as PointCast, BackWeb, and AirMedia have expressed support for the proposed standard. Such a standard raises the potential for push technology to be more widely integrated into the fabric of the Internet.

1.4 Sorting it All Out

The wide range of opinions on the pros and cons of push technology is understandable, given the fact that it is a major departure from the way distributed information systems have traditionally been built. Adding to the noise, however, is a wide-spread confusion about the basic principles of push and where it fits in to the world of data delivery. In this short paper we argue that this confusion stems from two fundamental causes: First, *push is just one dimension of a larger design space of data delivery mechanisms*. We identify three dimensions for data delivery mechanisms (push vs. pull is one of them) and show how different choices along these dimensions interact. Second, *networked information systems can employ different data delivery options between different sets of information producers and consumers*. Thus, complex systems will likely contain mixtures of push and pull (along with the other options) at various points in the network. In such a situation, it is inappropriate to identify an entire system as being "push-based" or "pull-based".

In the following, we present an overview of our ideas on data dissemination in order to provide a framework for thinking about push technology in the larger context of networked information systems. Our intent is to clarify some of the issues surrounding push technology and to characterize the design space for data delivery in dissemination-based information systems and applications.

2 Fundamental Properties

In this section, we present an overview of data delivery, focusing on how the notion of data push fits in with the other dimensions of the design space for delivery mechanisms. We then describe why it is often inappropriate to refer to complex distributed systems as simply "push-based" or "pull-based". A more detailed discussion of these issues can be found in [Fran97].

2.1 Options for Data Delivery

Support for different styles of data delivery allows a distributed information system to be optimized for various server, client, network, data, and application properties. We have identified three main characteristics that can be used to compare data delivery mechanisms: (1) push vs. pull; (2) periodic vs. aperiodic; and (3) unicast vs. 1-to-N. While there are numerous other dimensions that should be considered, such as fault-tolerance, ordering guarantees, error properties, network topology, etc., we have found that these three characteristics provide a good initial basis for discussing many popular approaches. In particular, we argue that all three of these characteristics must be considered in order to make intelligent choices about delivery mechanisms for specific situations. Figure 1 shows these characteristics and how several common mechanisms relate to them.

2.1.1 *Client Pull vs. Server Push*

We first focus on push vs. pull. Current database servers and object repositories manage data for clients that explicitly request data when they require it. When a request is received at a server, the server locates the information of interest and returns it to the client. This *request-response* style of operation is *pull-based* — the transfer of information from servers to clients is initiated by a client pull. In contrast, as discussed in the introduction, push-based data delivery involves sending information to a client population in advance of any specific request. With push-based delivery, the server initiates the transfer.

2.1.2 *Aperiodic vs. Periodic*

Both push and pull can be performed in either an aperiodic or periodic fashion. Aperiodic delivery is *event-driven* — a data request (for pull) or transmission (for push) is triggered by an event such as a user action (for pull) or data update (for push). In contrast, periodic delivery is performed according to some pre-arranged schedule. This schedule may be fixed, or may be generated with some degree of randomness.[2] An application that sends out stock prices on a regular basis is an example of periodic push, whereas one that sends out stock prices only when they change is an example of aperiodic push.

2.1.3 *Unicast vs. 1-to-N*

The third characteristic of data delivery mechanisms is whether they are based on unicast or 1-to-N communication. With unicast communication, data items are sent from a data source (e.g., a single server) to one other machine, while 1-to-N communication allows multiple machines to receive the data sent by a data source.[3]

Two types of 1-to-N data delivery can be distinguished: multicast and broadcast. With multicast, data is sent to a specific subset of clients who have indicated their interest in receiving the data. Since the recipients are known, given a two-way communications medium it is possible to make multicast reliable; that is, network protocols can be developed that guarantee the eventual delivery of the message to all clients that should receive it. In contrast, broadcasting sends information over a medium on which an unidentified and possibly unbounded set of clients can listen.

2.2 Classification of Delivery Mechanisms

It is possible to classify many existing data delivery mechanisms using the characteristics described above. Such a classification is shown in Figure 1. We discuss several of the mechanisms below.

Aperiodic Pull - Traditional request/response mechanisms use aperiodic pull over a unicast connection. If instead, a 1-to-N connection is used, then clients can "snoop" on the requests made by other clients, and obtain data that they haven't explicitly asked for (e.g, see [Acha97, Akso98]).

Periodic Pull - In some applications, such as remote sensing, a system may periodically send requests to other sites to obtain status information or to detect changed values. If the information is returned over a 1-to-N link, then as with request/response, other clients can snoop to obtain data items as they go by. Most existing Web or Internet-based "push" systems are actually implemented using Periodic Pull between the client machines and the data source(s).

Aperiodic Push - Publish/subscribe protocols are becoming a popular way to disseminate information in a network [Oki93, Yan95, Glan96]. In a publish/subscribe system, users provide information (sometimes in the form of a

[2] For the purposes of this discussion, we do not distinguish between fixed and randomized schedules. Such a distinction is important in certain applications. For example, algorithms for conserving energy in mobile environments proposed by Imielinski et al. [Imie94] depend on a strict schedule to allow mobile clients to "doze" during periods when no data of interest to them will be broadcast.

[3] Some systems attempt to implement a 1-to-N style of data delivery using unicast (i.e., by sending identical, individual messages to multiple clients). As discussed in Section 3, this type of pseudo-broadcast can result in tremendous bandwidth and server overload problems. For this reason, we classify such systems as "unicast-based" in our taxonomy.

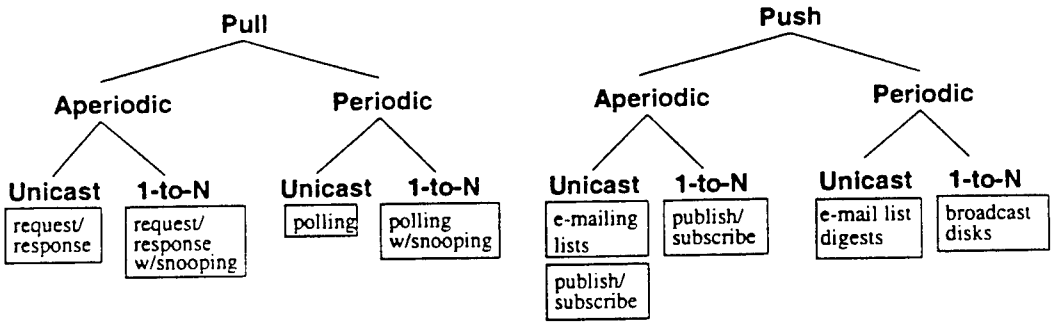


Figure 1: Data Delivery Options

profile) indicating the types of information they wish to receive. Publish/subscribe is push-based; data flow is initiated by the data sources, and is aperiodic, as there is no predefined schedule for sending data. Publish/subscribe protocols are inherently 1-to-N in nature, but due to limitations in current Internet technology, they are often implemented using individual unicast messages to multiple clients. Examples of such systems include Internet e-mail lists and some existing "push" systems on the Internet. True 1-to-N delivery is possible through technologies such as IP-Multicast, but such solutions are typically limited to individual Intranets or Local Area Networks.

Periodic Push - Periodic push has been used for data dissemination in many systems. An example of Periodic Push using unicast is Internet mailing lists that send out "digests" on a regular schedule. For example, the Majordomo system allows a list manager to set up a schedule (e.g., weekly) for sending digests. Such digests allow users to follow a mailing list without being continually interrupted by individual messages. There have also been many systems that use Periodic Push over a broadcast or multicast link. These include Tele-Text [Amma85, Wong88], DataCycle [Herm87], Broadcast Disks [Acha95a, Acha95b] and mobile databases [Imie94].

2.3 End-to-End Considerations

The second source of confusion about push technology is the fact that networked information systems typically contain many interconnected nodes. These nodes may be (logically) organized in various structures, and different data delivery mechanisms may be used between different sets of nodes. Given the potential heterogeneity of delivery mechanisms in a complex system, it is often not appropriate to describe the entire end-to-end (i.e., data source to consumer) system as "push-based" or "pull-based".

In general, a distributed information system can be though of as having three types of nodes: (1) *data sources*, which provide the base data that is to be disseminated; (2) *clients*, which are net consumers of information; and (3) *information brokers*, (or agents, mediators, etc.) that acquire information from other sources, add value to that information (e.g., some additional computation or organizational structure) and then distribute this information to other consumers. By creating hierarchies of brokers, information delivery can be tailored to the needs of many different users.

While the previous discussion has focused primarily on different modes of data delivery, the brokers provide the glue that binds these modes together. In many cases, the expected usage patterns of the brokers can drive the selection of which mode of delivery to use. For example, a broker that typically is very heavily loaded with requests could be an excellent candidate for a push-based delivery mechanism to its clients.

As we move upstream in the data delivery chain, brokers look like data sources to their clients. Receivers of information cannot detect the details of interconnections any further upstream than their immediate predecessor. This principle of *network transparency* allows data delivery mechanisms to change without having global impact. Suppose that node B is pulling data values from node A on demand. Further, suppose that node C is listening to a periodic broadcast from node B which includes values that B has pulled from A. Node C will not have to change it's data gathering strategy if A begins to push values to B. Changes in links are of interest only to the nodes that are directly involved. Likewise, this transparency allows the "appearance" of the data delivery at any node to differ from the way the data is actually delivered earlier in the network. This ability to change the appearance of data delivery is at the root of much of the confusion surrounding push technology.

Figure 2 shows a simple example of the importance of considering multiple network components and the impact of transparency. The figure shows how data delivery is performed in the initial versions of PointCast. To the user sitting at the screen, the system appears to be "push-based"; data flows across the screen without any user intervention. Due to current limitations of the Internet, however, that data is actually brought over to the client machine using a stream of periodic pull requests, delivered in a unicast fashion. Thus, the implementation of PointCast 1.0 between the client and the PointCast server is actually the exact opposite of the view that is presented to the user *in all three dimensions* of the hierarchy of Figure 1. This situation is not unique to PointCast;

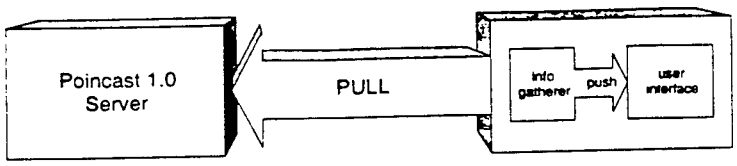


Figure 2: Pointcast 1.0

in fact, it is true for virtually all of the Internet-based push solutions, and stems from the fact that current IP and HTTP protocols do not adequately support push or 1-to-N communication.

3 Reexamining Current Push Technology

The previous section identified several of the sources of confusion in the current discussions and debate regarding push technology. In particular, the confusion stems from the mismatch between the user's perception and the actual data delivery mechanisms used by the system. Furthermore, this mismatch is also at the root of many of the performance concerns (particularly bandwidth overload) associated with current push technology. The impact of the mismatch on performance can be summarized as follows:

Pull instead of push - Current webcasting solutions typically use data pull to obtain information from data sources. This choice is due to limitations of the HTTP protocol, which is primarily pull-based. As stated previously, replacing push with pull requires that the pull be done in a *polling* manner. Polling can be quite resource intensive because it generates many requests. These requests consume client, server, and network resources. The problems are exacerbated if all clients poll individually, which could result in servers becoming overloaded due to the high volume of requests.

Periodic instead of aperiodic - Polling is typically done in a periodic manner that is independent of the events (e.g., data modifications) that would require data to be transfered. This independence results in a granularity problem: if polling is done too frequently, then the overhead can become substantial; if it is done too infrequently, then clients may unknowingly be accessing stale data.

Unicast instead of 1-to-N - In the absence of a true broadcast or multicast facility, systems that require 1-to-N behavior must implement it using multiple identical messages, one for each intended recipient. The potential bandwidth problems of such an approach are obvious. If n clients are interested in the same data item, then that same item must be sent over the network n times.

Fortunately, the concept of Network Transparency can be used to ameliorate this situation. One solution involves placing a local server inside an organization's firewall. All the clients interact with the local server in the way that is most appropriate for the local network and system configuration. The local server can then perform polling of the remote data source on behalf of the entire organization, which reduces Internet traffic. Likewise, the data source needs only to send a single copy of each data item to the local server, which can then distribute it to all the clients it represents. The local server can then multicast the data to its clients, if such capability exists.

4 Conclusions

In summary, push is currently a hot topic, but it is essential that it be placed in the proper context. Push is one choice (among many) for data delivery in distributed information systems. Push is not, for example, the same as broadcast. In fact, many existing push-based products are based on periodic pull over unicast connections. In our work on data dissemination, we have advocated a new look at the construction of distributed information systems that allows a seamless integration of all data delivery mechanisms including, but not limited to the various forms of push. We believe that this is a fertile area of work for the database community since the use of careful data management techniques in this context can have a significant impact on overall system performance and usability.

References

[Acha95a] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", *ACM SIGMOD Conf.*, San Jose, May, 1995.

[Acha95b] S. Acharya, M. Franklin, S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks", *IEEE Personal Communications*, 2(6), December, 1995.

[Acha97] S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Data Broadcast", *ACM SIGMOD Conf.*, 1997.

[Akso98] D. Aksoy, M. Franklin, "Scheduling for Large-Scale On-Demand Data Broadcasting", *Proc. IEEE INFOCOM Conf.*, San Francisco, March, 1998.

[Amma85] M. Ammar, J. Wong, "The Design of Teletext Broadcast Cycles", *Perf. Evaluation*, 5 (1985).

[Fran97] M. Franklin, S. Zdonik, "A Framework for Scalable Dissemination-Based Information Systems" *ACM OOPSLA Conf.*, Atlanta, October, 1997.

[Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communication", *CACM*, 33(2), February, 1990.

[Glan96] D. Glance, "Multicast Support for Data Dissemination in OrbixTalk", *IEEE Data Engineering Bulletin*, 19(3), Sept., 1996.

[Herm87] G. Herman, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", *Proc. ACM SIGMOD Conf.*, San Francisco, CA, May, 1987.

[Imie94] T. Imielinski, S. Viswanathan, B. Badrinath, "Energy Efficient Indexing on Air", *ACM SIGMOD Conf.*, 1994.

[Oki93] B. Oki, M. Pfluegl, A. Siegel, D. Skeen, "The Information Bus - An Architecture for Extensible Distributed Systems", *Proc. 14th SOSP*, Ashville, NC, December, 1993.

[Wong88] J. Wong, "Broadcast Delivery", *Proceedings of the IEEE*, 76(12), December, 1988.

[Yan95] T. Yan, H. Garcia-Molina, "SIFT – A Tool for Wide-area Information Dissemination", *Proc. 1995 USENIX Technical Conference*, 1995.

☆U.S. GOVERNMENT PRINTING OFFICE 2000-510-079-81243

WILLIAM E. RZEPKA AFRL/IFTD 525 BROOKS ROAD ROME, NY 13441-4505	5	
BROWN UNIVERSITY COMPUTER SCIENCE DEPT BOX 1910 PROVIDENCE, RI 02912	5	
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1	2
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	1	4
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1	7
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1	12
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1	17
AFRL/HESC-TDC 2698 G STREET, BLDG 190 WRIGHT-PATTERSON AFB OH 45433-7604	1	22
ATTN: SMDC IM PL US ARMY SPACE & MISSILE DEF CMD P.O. BOX 1500 HUNTSVILLE AL 35807-3801	1	24
COMMANDER, CODE 4TL000D TECHNICAL LIBRARY, NAWC-WD 1 ADMINISTRATION CIRCLE CHINA LAKE CA 93555-6100	1	27

CDR, US ARMY AVIATION & MISSILE CMD REDSTONE SCIENTIFIC INFORMATION CTR ATTN: AMSAM-RD-OB-R, (DOCUMENTS) REDSTONE ARSENAL AL 35898-5000	2	31
REPORT LIBRARY MS P364 LOS ALAMOS NATIONAL LABORATORY LOS ALAMOS NM 87545	1	33
ATTN: DEBORAH HART AVIATION BRANCH SVC 122.10 FOB10A, RM 931 800 INDEPENDENCE AVE, SW WASHINGTON DC 20591	1	36
AFIWC/MSY 102 HALL BLVD, STE 315 SAN ANTONIO TX 78243-7016	1	38
ATTN: KAROLA M. YOURISON SOFTWARE ENGINEERING INSTITUTE 4500 FIFTH AVENUE PITTSBURGH PA 15213	1	39
USAF/AIR FORCE RESEARCH LABORATORY AFRL/VSOSA(LIBRARY-BLDG 1103) 5 WRIGHT DRIVE HANSCOM AFB MA 01731-3004	1	44
ATTN: EILEEN LADUKE/D460 MITRE CORPORATION 202 BURLINGTON RD BEDFORD MA 01730	1	45
OUSD(P)/DTSA/DUTD ATTN: PATRICK G. SULLIVAN, JR. 400 ARMY NAVY DRIVE SUITE 300 ARLINGTON VA 22202	1	46
SOFTWARE ENGR'G INST TECH LIBRARY ATTN: MR DENNIS SMITH CARNEGIE MELLON UNIVERSITY PITTSBURGH PA 15213-3890	1	94
USC-ISI ATTN: DR ROBERT M. BALZER 4676 ADMIRALTY WAY MARINA DEL REY CA 90292-6695	1	277

KESTREL INSTITUTE ATTN: DR CORDELL GREEN 1801 PAGE MILL ROAD PALO ALTO CA 94304	1	313
ROCHESTER INSTITUTE OF TECHNOLOGY ATTN: PROF J. A. LASKY 1 LOMB MEMORIAL DRIVE P.O. BOX 9887 ROCHESTER NY 14613-5700	1	410
AFIT/ENG ATTN:TOM HARTRUM WPAFB OH 45433-6583	1	565
THE MITRE CORPORATION ATTN: MR EDWARD H. BENSLEY BURLINGTON RD/MAIL STOP A350 BEDFORD MA 01730	1	568
ANDREW A. CHIEN SAIC CHAIR PROF (SCI APL INT CORP) USCD/CSE-AP&M 4308 9500 GILMAN DRIVE, DEPT. 0114 LAJOLLA CA 92093-0114	1	573
HONEYWELL, INC. ATTN: MR BERT HARRIS FEDERAL SYSTEMS 7900 WESTPARK DRIVE MCLEAN VA 22102	1	574
SOFTWARE ENGINEERING INSTITUTE ATTN: MR. WILLIAM E. HEFLEY CARNEGIE-MELLON UNIVERSITY 304 OAK GROVE CT WESFORD PA 15090	1	576
UNIVERSITY OF SOUTHERN CALIFORNIA ATTN: DR. YIGAL ARENS INFORMATION SCIENCES INSTITUTE 4676 ADMIRALTY WAY/SUITE 1001 MARINA DEL REY CA 90292-6695	1	577
COLUMBIA UNIV/DEPT COMPUTER SCIENCE ATTN: DR GAIL E. KAISER 450 COMPUTER SCIENCE BLDG 500 WEST 120TH STREET NEW YORK NY 10027	1	578
AFIT/ENG ATTN: DR GARY B. LAMONT SCHOOL OF ENGINEERING DEPT ELECTRICAL & COMPUTER ENGRG WPAFB OH 45433-6583	1	582

NSA/OFC OF RESEARCH ATTN: MS MARY ANNE OVERMAN 9800 SAVAGE ROAD FT GEORGE G. MEADE MD 20755-6000	1	584
TEXAS INSTRUMENTS INCORPORATED ATTN: DR DAVID L. WELLS P.O. BOX 655474, MS 238 DALLAS TX 75265	1	593
KESTREL DEVELOPMENT CORPORATION ATTN: DR RICHARD JULLIG 3260 HILLVIEW AVENUE PALO ALTO CA 94304	1	774
DARPA/ITO ATTN: DR KIRSTIE BELLMAN 3701 N FAIRFAX DRIVE ARLINGTON VA 22203-1714	1	1001
NASA/JOHNSON SPACE CENTER ATTN: CHRIS CULBERT MAIL CODE PT4 HOUSTON TX 77058	1	1004
STERLING IMD INC. KSC OPERATIONS ATTN: MARK MAGINN BEECHES TECHNICAL CAMPUS/RT 26 N. ROME NY 13440	1	1058
SCHLUMBERGER LABORATORY FOR COMPUTER SCIENCE ATTN: DR. GUILLERMO ARANGO 8311 NORTH FM620 AUSTIN, TX 78720	1	1326
DECISION SYSTEMS DEPARTMENT ATTN: PROF WALT SCACCHI SCHOOL OF BUSINESS UNIVERSITY OF SOUTHERN CALIFORNIA LOS ANGELES, CA 90089-1421	1	1329
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY ATTN: CHRIS DABROWSKI ROOM A266, BLDG 225 GAITHSBURG MD 20899	1	1333
EXPERT SYSTEMS LABORATORY ATTN: STEVEN H. SCHWARTZ NYNEX SCIENCE & TECHNOLOGY 500 WESTCHESTER AVENUE WHITE PLAINS NY 20604	1	1334

NAVAL TRAINING SYSTEMS CENTER ATTN: ROBERT BREAUX/CODE 252 12350 RESEARCH PARKWAY ORLANDO FL 32826-3224	1	1335
DR JOHN SALASIN DARPA/ITO 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1	1634
DR BARRY BOEHM DIR, USC CENTER FOR SW ENGINEERING COMPUTER SCIENCE DEPT UNIV OF SOUTHERN CALIFORNIA LOS ANGELES CA 90089-0781	1	1639
DR STEVE CROSS CARNEGIE MELLON UNIVERSITY SCHOOL OF COMPUTER SCIENCE PITTSBURGH PA 15213-3891	1	1640
DR MARK MAYBURY MITRE CORPORATION ADVANCED INFO SYS TECH; G041 BURLINTON ROAD, M/S K-329 BEDFORD MA 01730	1	1641
ISX ATTN: MR. SCOTT FOUSE 4353 PARK TERRACE DRIVE WESTLAKE VILLAGE,CA 91361	1	1642
MR GARY EDWARDS ISX 433 PARK TERRACE DRIVE WESTLAKE VILLAGE CA 91361	1	1643
LEE ERMAN CIMFLEX TEKNOWLEDGE 1810 EMBACADERO ROAD P.O. BOX 10119 PALO ALTO CA 94303	1	1647
DR. DAVE GUNNING DARPA/ISO 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1	1669
DR. MICHAEL PITTARELLI COMPUTER SCIENCE DEPART SUNY INST OF TECH AT UTICA/ROME P.O. BOX 3050 UTICA, NY 13504-3050	1	1815

CAPPARO TECHNOLOGIES, INC ATTN: GERARD CAPPARO 311 TURNER ST. UTICA, NY 13501	1	1816
USC/ISI ATTN: BOB MCGREGOR 4676 ADMIRALTY WAY MARINA DEL REY, CA 90292	1	1817
SRI INTERNATIONAL ATTN: ENRIQUE RUSPINI 333 RAVENSWOOD AVE MENLO PARK, CA 94025	1	1818
DARTMOUTH COLLEGE ATTN: DANIELA RUS DEPT OF COMPUTER SCIENCE 11 ROPE FERRY ROAD HANOVER, NH 03755-3510	1	1819
UNIVERSITY OF FLORIDA ATTN: ERIC HANSON CISE DEPT 456 CSE GAINESVILLE, FL 32611-6120	1	1820
CARNEGIE MELLON UNIVERSITY ATTN: TOM MITCHELL COMPUTER SCIENCE DEPARTMENT PITTSBURGH, PA 15213-3890	1	1821
CARNEGIE MELLON UNIVERSITY ATTN: MARK CRAVEN COMPUTER SCIENCE DEPARTMENT PITTSBURGH, PA 15213-3890	1	1822
UNIVERSITY OF ROCHESTER ATTN: JAMES ALLEN DEPARTMENT OF COMPUTER SCIENCE ROCHESTER, NY 14627	1	1823
TEXTWISE, LLC ATTN: LIZ LIDDY 2-121 CENTER FOR SCIENCE & TECH SYRACUSE, NY 13244	1	1824
WRIGHT STATE UNIVERSITY ATTN: DR. BRUCE BERRA DEPART OF COMPUTER SCIENCE & ENGIN DAYTON, OHIO 45435-0001	1	1825

UNIVERSITY OF FLORIDA ATTN: SHARMA CHAKRAVARTHY COMPUTER & INFOR SCIENCE DEPART GAINESVILLE, FL 32622-6125	1	1826
KESTREL INSTITUTE ATTN: DAVID ESPINOSA 3260 HILLVIEW AVENUE PALO ALTO, CA 94304	1	1827
USC/INFORMATION SCIENCE INSTITUTE ATTN: DR. CARL KESSELMAN 11474 ADMIRALTY WAY, SUITE 1001 MARINA DEL REY, CA 90292	1	1829
MASSACHUSETTS INSTITUTE OF TECH ATTN: DR. MICHAELE SIEGEL SLOAN SCHOOL 77 MASSACHUSETTS AVENUE CAMBRIDGE, MA 02139	1	1830
USC/INFORMATION SCIENCE INSTITUTE ATTN: DR. WILLIAM SWARTHOUT 11474 ADMIRALTY WAY, SUITE 1001 MARINA DEL REY, CA 90292	1	1831
STANFORD UNIVERSITY ATTN: DR. GIO WIEDERHOLD 857 SIERRA STREET STANFORD SANTA CLARA COUNTY, CA 94305-4125	1	1832
SPAWARSYSCEN D44209 ATTN: LEAH WONG 53245 PATTERSON ROAD SAN DIEGO, CA 92152-7151	1	1833
SPAWAR SYSTEM CENTER D4123 ATTN: LES ANDERSON 53560 HULL STREET SAN DIEGO CA 92152	1	1834
GEORGE MASON UNIVERSITY ATTN: SUSHIL JAJODIA ISSE DEPT FAIRFAX, VA 22030-4444	1	1835
DIRNSA ATTN: MICHAEL R. WARE DOD, NSA/CSS (R23) FT. GEORGE G. MEADE MD 20755-6000	1	1836

DR. JIM RICHARDSON 3660 TECHNOLOGY DRIVE MINNEAPOLIS, MN 55418	1	1837
LOUISIANA STATE UNIVERSITY COMPUTER SCIENCE DEPT ATTN: DR. PETER CHEN 257 COATES HALL BATON ROUGE, LA 70803	1	1838
INSTITUTE OF TECH DEPT OF COMP SCI ATTN: DR. JAIDEEP SRIVASTAVA 4-192 EE/CS 200 UNION ST SE MINNEAPOLIS, MN 55455	1	1839
GTE/BBN ATTN: MAURICE M. MCNEIL 9655 GRANITE RIDGE DRIVE SUITE 245 SAN DIEGO, CA 92123	1	1840
UNIVERSITY OF FLORIDA ATTN: DR. SHARMA CHAKRAVARTHY E470 CSE BUILDING GAINESVILLE, FL 32611-6125	1	1841
AFRL/IFT 525 BROOKS ROAD ROME, NY 13441-4505	1	1865
AFRL/IFTM 525 BROOKS ROAD ROME, NY 13441-4505	1	1866
CENTRIC ENGINEERING SYSTEM, INC. 624 EAST EVELYN AVENUE SUNNYVALE, CA 94086-6483	1	1874
FLUENT INCORPORATED 500 DAVIS STREET, SUITE 600 EVANSTON, IL 60201	1	1875
THE MACNEAL-SCHWENDLER CORPORATION 815 COLORADO BOULEVARD LOS ANGELES, CA 90041-1777	1	1876

MOLECULAR SIMULATIONS, INC. 9865 SCRANTON ROAD SAN DIEGO, CA 92121-3752	1	1877
CENTRIC ENGINEERING SYSTEM, INC. 624 EAST EVELYN AVENUE SUNNYVALE, CA 94086-6488	1	1878
*Total Number of Copies is:	92	

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.